


Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное  
образовательное учреждение высшего образования  
«Сыктывкарский государственный университет имени Питирима Сорокина»  
(ФГБОУ ВО «СГУ им. Питирима Сорокина»)

Институт точных наук и информационных технологий  
Кафедра информационных систем


Курсовая работа по дисциплине «Базы данных»  
**Выбор города для посещения**

Направление подготовки  
09.03.03 Прикладная информатика  
Направленность (профиль) программы  
Прикладная информатика в экономике

Исполнитель:  
Гончаров Игорь Валерьевич

  
\_\_\_\_\_  
Личная подпись

Научный руководитель:  
Канд. педагогических наук, доцент  
Бабикова Надежда Николаевна

  
\_\_\_\_\_  
Личная подпись

Сыктывкар  
2021

## Оглавление

Аннотация .....	2
Введение .....	3
Анализ предметной области.....	4
1.1 Описание предметной области .....	4
1.2 Концептуальное моделирование предметной области .....	5
1.2.1 Функциональное моделирование .....	5
1.2.2 Моделирование данных .....	6
1.3 Обзор аналогичных информационных систем .....	9
Проектирование базы данных.....	10
2.1 Логическое проектирование БД .....	10
2.2 Физическое проектирование БД.....	13
Разработка программного обеспечения .....	23
3.1 Разработка пользовательских интерфейсов .....	23
3.2 Разработка программного обеспечения.....	29
3.3 Тестирование информационной системы.....	42
Заключение .....	45
Список источников .....	46
Тезаурус.....	47

## **Аннотация**

Целью данной курсовой работы является создание программного продукта с графическим пользовательским интерфейсом, служащего информационной системой, которая решает задачи выбора определенной страны и города в зависимости от предпочтений пользователя.

Для реализации программного продукта было решено использовать среды СУБД Access (создание БД) и язык Delphi (создание приложения с GUI).

Данный продукт нацелен на пользователей, выбирающих место для проведения своего отпуска.

Ключевые слова: данные, таблица, город, форма, страна, ключ, атрибут.

## **Введение**

Перед тем как отправиться в путешествие необходимо выбрать место отдыха. При этом необходимо учитывать различные факторы, являющиеся предпочтениями путешественника. Если не говорить о бюджете, многое в этом вопросе зависит от того, что человек хочет получить. Полный покой на фоне природы, активный и бодрящий отдых, оздоровление или культурную программу и так далее. В любом случае нужно иметь какую-либо информацию о потенциальных местах отдыха.

В качестве объекта курсовой работы выступает бизнес-процесс «выбор места отдыха». Предмет исследования – автоматизация бизнес-процесса.

Целью является создание приложения, в основе которого лежит специально спроектированная реляционная база данных, с помощью которого пользователь может выбрать город и страну для посещения на основе своих предпочтений.

### **Задачи исследования:**

- Анализ предметной области.
- Моделирование предметной области.
- Проектирование базы данных.
- Создание БД в среде Access.
- Создание приложения с GUI в среде Delphi.

## **Анализ предметной области**

### **1.1 Описание предметной области**

В данной курсовой работе производится разработка базы данных, в которой зафиксирована информация о странах и городах, расположенных в них. У городов и стран можно выделить большое число атрибутов, но в этой работе нас интересуют только те, на которые обычно смотрят люди, запланировавшие путешествие.

При возникновении желания путешествовать или просто отдохнуть, перед человеком встает выбор: куда ему отправиться. В интернете есть много сервисов, помогающих определиться с отелем, но они не предполагают помощь с выбором страны и города, а именно с этого было-бы логично начинать. Перед выбором отеля нужно прийти к выбору страны и города, подходящих вам. Как это сделать? Можно, конечно, спросить совета у знакомых, но нужно учитывать, что вкусы и увлечения у всех разные, и, даже если одному человеку определенный курорт понравился, он может совершенно не подойти другому путешественнику. Можно попробовать самому найти информацию и начать изучать статьи или видео в интернете, но, зачастую, они не несут полезной и уникальной информации, и вообще это занимает довольно много времени.

Видно, что поиск этой информации вручную неудобен, и в такой момент возникает идея автоматизировать этот процесс выбора: занести все популярные туристические направления в базу данных, задать им подходящие атрибуты: часть света, климат, сезоны отдыха, направленности отдыха, инфраструктура, размеры городов, отличие стран в культуре.

Это позволит быстро находить подходящее место.

## 1.2 Концептуальное моделирование предметной области

### 1.2.1 Функциональное моделирование

Моделируемая система представляет собой систему поиска городов и стран посредством выбора подходящих пользователю параметров.

Визуализация планируемой функциональности ИС в формате диаграммы «вариантов использования» (Use Case):



Рисунок 1 – Диаграмма Use Case

Требования к разрабатываемой системе:

- **Завершенность.** После реализации система не должна требовать доработок.

- **Актуальность.** Система не должна требовать обновления сразу после внедрения.
- **Обязательность.** Требования к ИС, обусловленные бизнес-процессами, должны быть учтены.

### 1.2.2 Моделирование данных

**Моделирование данных** – это процесс создания модели данных для хранения данных в базе данных. Эта модель данных представляет собой концептуальное представление объектов данных, связей между различными объектами данных и правилами.

**Концептуальная модель.** Основная цель этой модели – установить сущности, их атрибуты и их взаимосвязи. На этом уровне моделирования данных только основная информация о фактической структуре базы данных.

#### **Характеристики концептуальной модели данных:**

- Предлагает общеорганизационный охват бизнес-концепций.
- Этот тип моделей данных предназначен для бизнес-аудитории.
- Концептуальная модель разрабатывается независимо от технических характеристик оборудования, таких как емкость хранилища данных, расположение или спецификации программного обеспечения, таких как поставщик СУБД и технологии. Цель состоит в том, чтобы представлять данные так, как их увидит пользователь в «реальном мире».

3 основных части модели данных:

- Сущность (реальная вещь)
- Атрибут (характеристики или свойства объекта)

- Отношение (Зависимость или связь между двумя объектами)

Рассмотрим сущности нашей предметной области и их атрибуты:

1. Страна. Атрибуты – название, часть света, климат, направленность, рекомендуемый сезон, отличие в культуре.

- Название – наименование страны.
- Часть света – Африка, Ближний Восток, Америка, Азия и Океания, Европа.
- Климат – субтропический, тропический, средиземноморский, тропический пустынный, разнообразный, континентальный, экваториальный, субэкваториальный, умеренный.
- Направленность – культурно-экскурсионный, оздоровительный, активный, рекреационный.
- Рекомендуемый сезон – декабрь-февраль, март-май, июнь-август, сентябрь-ноябрь.
- Отличие в культуре – принимает значения «да» или «нет», подразумевается отличие от европейской культуры.

2. Город. Атрибуты – название, страна, инфраструктура, фото, размер, описание.

- Название – наименование города.
- Страна – страна, в которой расположен город.
- Инфраструктура – старый город, современный город, пляж, рынки/бутики, серфинг/дайвинг, кухня и рестораны, музеи, экскурсии, ночные клубы.
- Фото – изображение города.
- Размер – малый, средний, крупный, крупнейший.
- Описание – краткая информация о городе.



Сущности «Инфраструктура», «Направленность», «Сезоны» нужны для записи в них множества значений атрибутов «инфраструктура», «направленность» и «рекомендуемый сезон» сущностей «Город» и «Страна».

Кроме данных атрибутов все сущности имеют первичный ключ - атрибут, который однозначно определяет кортеж в отношении. Ключ служит как ограничение целостности в рамках одной таблицы для однозначной идентификации, поле первичного ключа не может повторяться или быть пустым.

Сейчас общепринятым является способ, при котором первичный ключ - это искусственный номер. Мы тоже так поступаем и вводим для сущностей атрибут ID.

Ниже представлена диаграмма «сущность – связь»:

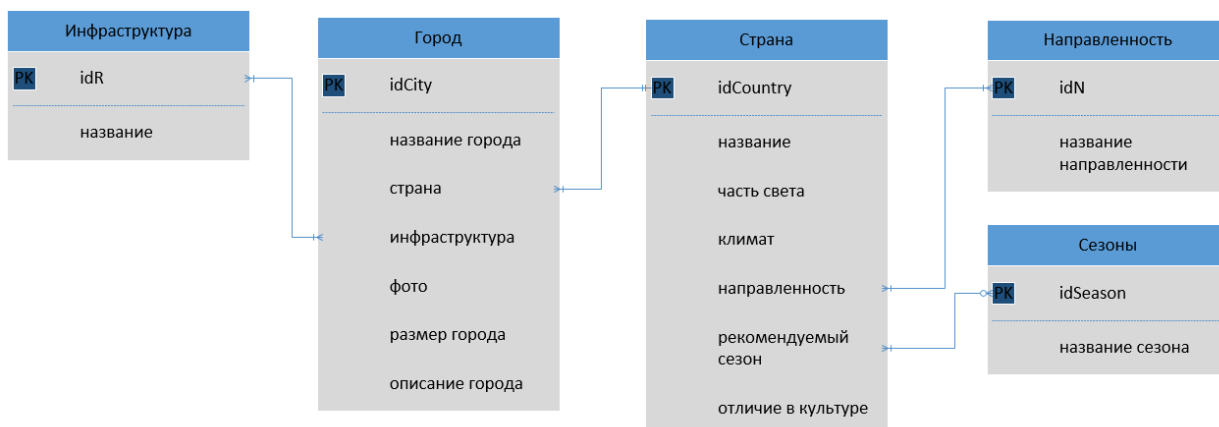


Рисунок 2 – Диаграмма «сущность – связь»

Отношения между сущностями могут быть нескольких типов: «один к одному», «один ко многим» и «многие ко многим» (при условии наличия двух сущностей, A и B):

- *Один ко многим.* В таких отношениях каждый экземпляр сущности A может иметь отношение с несколькими членами другой сущности B.

- *Один к одному.* В таких отношениях только один экземпляр любой из сущностей может иметь отношение с экземпляром другой сущности.
- *Многие ко многим.* В таких отношениях каждый экземпляр сущности А может иметь отношение с одним и более экземплярами сущности В, а каждый экземпляр сущности В – с одним и более экземплярами сущности А.

Страна и город связаны отношением «один-ко-многим». В типе связей один ко многим одной записи первой сущности соответствует несколько записей в другой сущности. Очевидно, что в одной стране находится множество городов.

Отношения «Инфраструктура» – «Город», «Направленность» – «Страна», «Сезоны» – «Страна» имеют тип связи «многие-ко-многим». Это обусловлено тем, что в полях сущностей «Город» и «Страна» может быть несколько значений. Например, у разных стран могут быть одновременно несколько разных подходящих для отдыха сезонов.

### **1.3 Обзор аналогичных информационных систем**

В процессе выполнения курсовой был работы проведен анализ автоматизированных систем поиска места проведения отдыха. Рассмотрены различные интернет ресурсы, посвященные вопросам автоматизации поиска городов.

Анализ аналогичных систем дал следующие результаты: на момент написания не найдены аналогичные информационные системы по теме курсовой работы. Найденные системы подбирают, как правило, только отели и рассматривают их атрибуты, не помогая в выборе страны и города.

Поэтому задача создания данной ИС является актуальной.

# Проектирование базы данных

## 2.1 Логическое проектирование БД

**Логическое проектирование базы данных** – это процесс создания модели используемой информации на основе выбранной модели организации данных, но без учета типа целевой СУБД и других физических аспектов реализации.

Концептуальная модель данных, созданная на предыдущем этапе, уточняется и преобразуется в логическую модель данных. Логическая модель данных учитывает особенности выбранной модели организации данных в целевой СУБД (в нашем случае реляционная модель БД).

**Реляционная база данных** – это упорядоченная информация, связанная между собой определёнными отношениями. Логически такая база данных представлена в виде таблиц, в которых и лежит вся эта информация.

На этапе логического проектирования реляционной БД необходимо избавиться от связи «многие ко многим» в отношениях. В нашем случае мы должны устранить связь «многие ко многим» между отношениями «город» – «инфраструктура», «страна» – «направленность», «страна» – «сезоны». Для этого добавим соединительные отношения.

Рассмотрим на примере «город-инфраструктура»:



Рисунок 3 – Связь «многие ко многим»

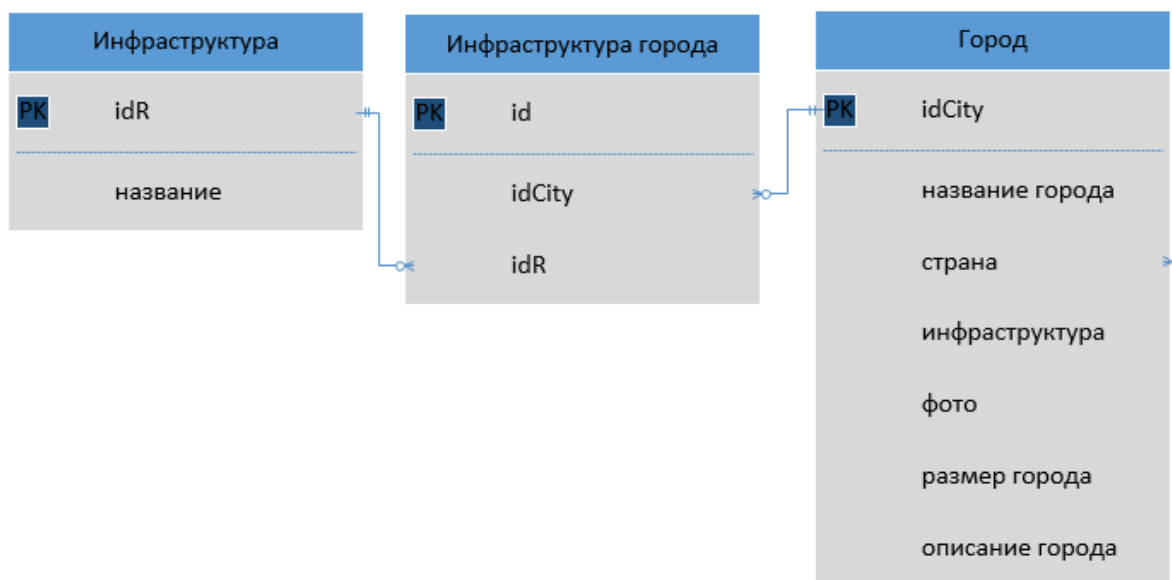


Рисунок 4 – Добавляем соединительную таблицу

Теперь в отношении «Инфраструктура города» мы можем хранить повторяющиеся ключи «idCity» отношения «Город» и «idR» отношения «Инфраструктура». Связь с основными отношениями устанавливается по ключевому полю, и теперь она вида «один ко многим».

Повторим это действие для отношений «страна», «направленность», «сезоны»: создадим соединительные отношения «Направленность в стране» и «Рекомендуемый сезон», в которых будут храниться повторяющиеся ключи «idCountry» отношения «Страна», «idN» и «idSeason» отношений «Направленность» и «Сезоны».

В итоге логическая модель примет следующий вид:

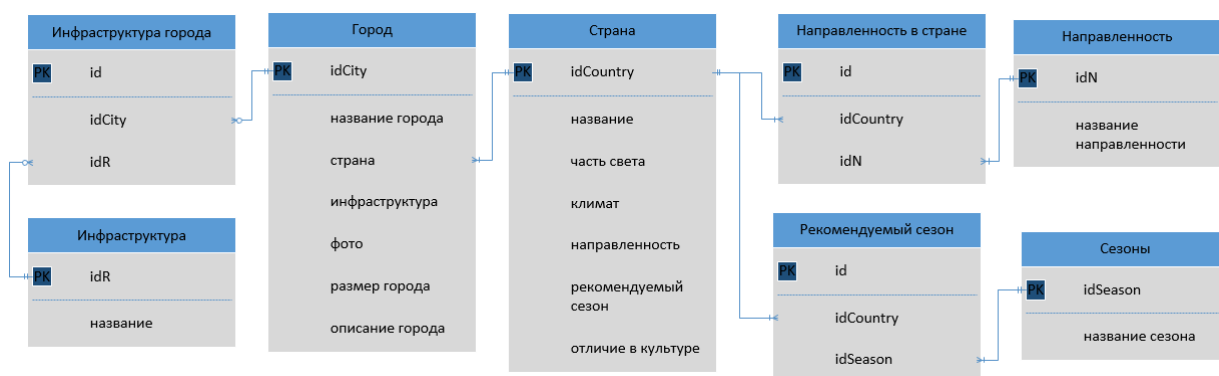


Рисунок 5 – Логическая модель данных

Созданная логическая модель данных является основным источником информации для этапа физического проектирования. Логическая модель данных имеет также важную роль на этапе эксплуатации и сопровождения уже готовой системы. При правильно организованном сопровождении поддерживаемая в актуальном состоянии модель данных позволяет точно и наглядно представить любые вносимые в базу данных изменения, а также оценить их влияние на прикладные программы и использование данных, уже имеющихся в базе.

## 2.2 Физическое проектирование БД

**Физическое проектирование базы данных** представляет собой процесс подготовки описания реализации этой базы во вторичной памяти. Создается описание таблиц базы данных и выбранных для них структур хранения, а также методов доступа, которые будут использоваться для эффективного доступа к данным.

Для выполнения данной курсовой работы была предложена СУБД Microsoft Access. Access является реляционной СУБД, которая поддерживает все средства и возможности по обработке данных, свойственные реляционным моделям. Данные, которые необходимо хранить в базе данных, могут быть представлены в различных форматах, в частности, денежном, графическом, числовом и других. СУБД MS Access обычно применяют, когда задача требует хранения и обработки различного рода информации о большом количестве объектов. Также в данном пакете предусмотрена защита, позволяющая повысить сохранность данных, находящихся в базе данных.

Физическое проектирование БД базируется на принципах нормализации. Процесс нормализации базы данных выглядит следующим образом: мы, следуя определённым правилам и соблюдая определенные требования, проектируем таблицы в базе данных. При этом все эти правила и требования можно сгруппировать в несколько наборов, и если спроектировать базу данных с соблюдением всех правил и требований, которые включаются в тот или иной набор, то база данных будет находиться в определённом состоянии, т.е. форме, и такая форма называется нормальная форма базы данных.

**Нормальная форма** – свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, потенциально приводящей к логически ошибочным результатам выборки или изменения данных. Нормальная форма определяется как совокупность требований,

которым должно удовлетворять отношение. Конечной целью нормализации является уменьшение потенциальной противоречивости хранимой в базе данных информации. **Избыточность данных** – одни и те же данные хранятся в базе в нескольких местах, это приводит к аномалиям.

Общее назначение процесса нормализации заключается в следующем:

- исключение некоторых типов избыточности
- устранение некоторых аномалий обновления
- разработка проекта базы данных, который является достаточно «качественным» представлением реального мира, интуитивно понятен и может служить хорошей основой для последующего расширения
- упрощение процедуры применения необходимых ограничений целостности

Избыточность устраняется, как правило, за счёт декомпозиции отношений, то есть разбиения одного отношения на несколько.

Существует 5 основных нормальных форм базы данных:

- Первая нормальная форма (1NF)
- Вторая нормальная форма (2NF)
- Третья нормальная форма (3NF)
- Четвертая нормальная форма (4NF)
- Пятая нормальная форма (5NF)

Каждая следующая нормальная форма содержит более строгие правила и критерии, следовательно, приводя базу данных к определённой нормальной форме мы устраняем определённый набор аномалий. Отсюда можно сделать вывод, что чем выше нормальная форма, тем меньше аномалий в базе будет. База данных считается нормализованной, если она находится как минимум в

третьей нормальной форме (3NF). Как правило, нормализация до третьей нормальной формы (3NF) является обычной, стандартной практикой, так как 3NF устраняет достаточное количество аномалий, при этом производительность базы данных, а также удобство ее использования не снижается, что нельзя сказать о всех последующих формах.

Ситуации, при которых требуется нормализовать базу данных до четвертой нормальной формы (4NF) встречаются достаточно редко, поэтому мы и остановимся на 3NF.

### **Проведем нормализацию БД до 3NF**

Перед тем как переходить к процессу приведения таблиц базы данных к первой нормальной форме, необходимо чтобы эти таблицы соблюдали базовые принципы реляционной теории - строки в таблицах не должны быть пронумерованы, порядок строк не имеет значения, так же как не имеет значения порядок столбцов. Данных в нашей БД еще нет, следовательно, никакой нумерации тоже нет, принципы реляционной теории соблюдены.

*Приведение к 1NF.* Первая нормальная форма - это обычное отношение. Отношение в 1NF обладает следующими свойствами:

- В отношении нет одинаковых кортежей.
- Кортежи не упорядочены.
- Атрибуты не упорядочены.
- Все значения атрибутов атомарны (имеют единственное значение).

Это можно учесть при проектировании, наложив ограничения целостности, такие как длина поля, тип поля и дополнительные условия.



*Приведение к 2NF.* Если в некоторых отношениях обнаружена зависимость атрибутов от части сложного ключа, то проводится декомпозиция этих отношений на несколько отношений следующим образом: те атрибуты, которые зависят от части сложного ключа, выносятся в отдельное отношение вместе с этой частью ключа. В исходном отношении остаются все ключевые атрибуты. Требования второй нормальной формы:

- Таблица должна находиться в первой нормальной форме.
- Таблица должна иметь ключ.
- Все неключевые атрибуты таблицы должны зависеть от полного ключа (в случае если он составной).

Таблица уже находится в первой нормальной форме, ключи мы задали еще на стадии концептуального проектирования, а составных ключей в нашей БД нет.

*Приведение к 3NF.* Требование третьей нормальной формы – отсутствие в таблицах транзитивной зависимости. Транзитивная зависимость – зависимость неключевых атрибутов от значений других неключевых атрибутов.

Если в некоторых отношениях обнаружена зависимость некоторых неключевых атрибутов от других неключевых атрибутов, в свою очередь зависящих от ключа, то проводится декомпозиция этих отношений следующим образом: те неключевые атрибуты, которые зависят от других неключевых атрибутов, выносятся в отдельное отношение. В новом отношении ключом становится детерминант функциональной зависимости.

В нашем случае в таблице «город» имеется атрибут «размер города». Он принимает символьные значения вида «малый», «средний» и т.д. Проблема в том, что эти значения объективно не отражают размер города,

так как для каждого потенциального клиента эти слова могут восприниматься по-разному. Возникает необходимость добавить атрибут «описание размера города». Добавление этого атрибута в таблицу «город» будет ошибкой, таблица не будет подходить под критерии 3NF, так как атрибут «описание размера города» не связан напрямую с городом, он связан напрямую со столбцом «размер города», который напрямую связан с городом. Это и есть транзитивная зависимость. Решается эта проблема путем декомпозиции. Разбиваем таблицу на две: «город» и «размеры городов». В таблице «размеры городов» будем хранить название размера и его описание, а в таблице «город» внешний ключ, для связи с таблицей «размеры городов»:

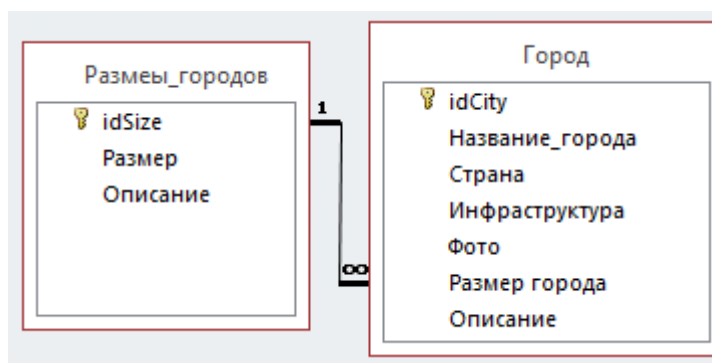


Рисунок 6 – Пример приведения к 3NF

## Справочники

Справочник в БД – это таблица, которая используется для работы со списками данных. Как правило, информация в справочник вводится единожды и не требует изменений или дополнений. Справочник создается в таких случаях, когда атрибут в таблице может принимать фиксированный набор значений.

В данной БД такими атрибутами являются «часть света» и «климат» таблицы «страна». Создадим для них одноименные таблицы и свяжем их с аналогичными атрибутами таблицы «страна»:

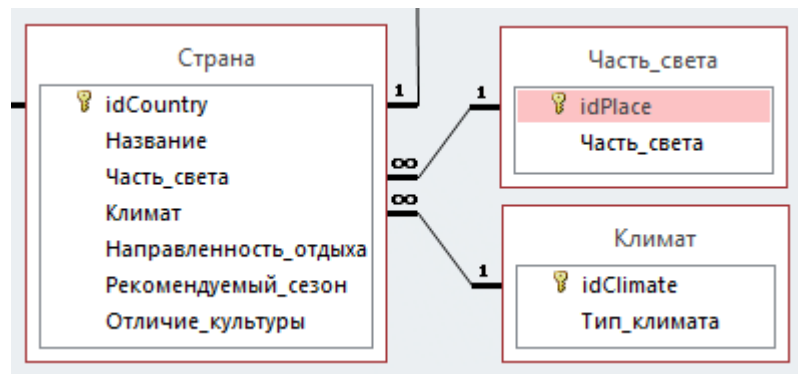


Рисунок 7 – Создание справочников

Имея такие таблицы в БД Access мы можем сделать подстановку для полей «часть света» и «климат», что упростит нам заполнение таблицы. Кроме того, это послужит дополнительным ограничением целостности БД.

### Ограничения целостности

Целостность базы данных – соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным правилам. Каждое правило, налагающее некоторое ограничение на возможное состояние базы данных, называется ограничением целостности.

Ограничения целостности можно определить, как специальные средства в базах данных, главное назначение которых – не дать попасть в базу недопустимым данным (например, предупредить ошибки пользователей при вводе данных). Все ограничения целостности можно разделить на три большие категории:

*1. Средства обеспечения доменной целостности.* Они отвечают за то, чтобы в соответствующем поле базы данных были допустимые значения. Например, фамилия, как правило, должна состоять из букв, а почтовый индекс – из цифр. В Access за это отвечает тип данных поля.

2. *Сущностная целостность*. Главная задача здесь – сделать так, чтобы данные об одной сущности не попали в базу данных два раза. Обеспечивается ограничением уникальности и первичным ключом;

3. *Ссылочная целостность*. обеспечивается системой первичных и внешних ключей. Например, при помощи этих средств можно гарантировать, что у нас не будет городов, находящихся в странах, которых нет в базе данных. Ссылочная целостность в Access осуществляется при создании схемы данных:

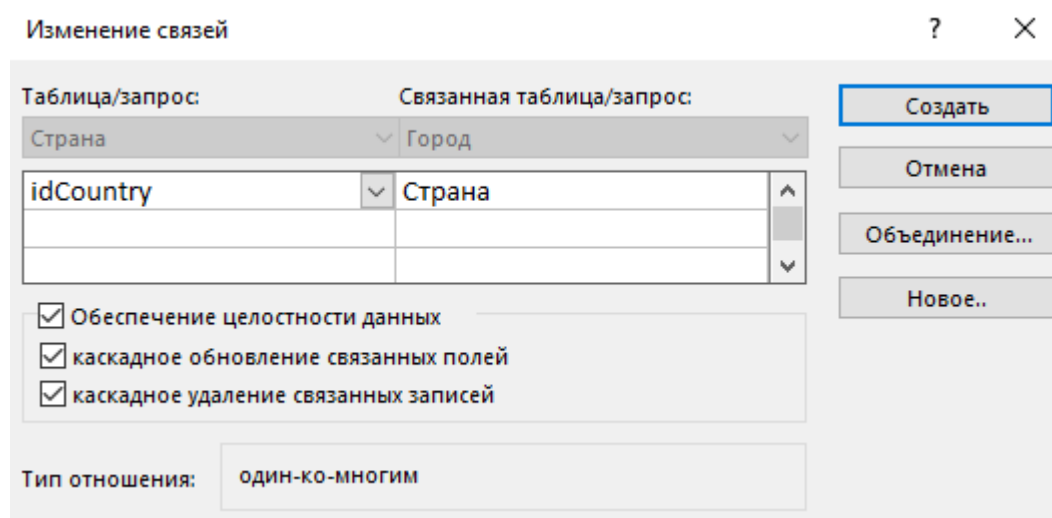


Рисунок 8 – Обеспечение целостности

Схема данных в Access является не только средством графического отображения логической структуры базы данных, она активно используется системой в процессе обработки данных. Создание схемы данных позволяет упростить конструирование многотабличных форм, запросов, отчетов.

### **Каскадное обновление и удаление связанных записей**

Если для выбранной связи обеспечивается поддержание целостности, можно задать режим каскадного удаления связанных записей и режим каскадного обновления связанных полей. Такие параметры делают

возможным в главной таблице, соответственно, удаление записей и изменение значения в ключевом поле, так как при этих параметрах система автоматически выполнит необходимые изменения в подчиненных таблицах, обеспечив сохранение свойств целостности базы данных.

В Access установлены следующие ограничения:

- связанное поле главной таблицы является ключевым полем или имеет уникальный индекс
- связанные поля имеют один тип данных
- если таблицы являются связанными, то они должны быть таблицами базы данных Access. Для связанных таблиц из баз данных других форматов установить целостность данных невозможно

Перенесем нашу логическую модель данных в среду СУБД Microsoft Access.

### Структура таблиц Access:

Название таблицы	Имя поля	Тип данных, длина	Примечание
Страна	idCountry	Счетчик	первичный ключ
	Название	Короткий текст (30)	обязательное поле
	Часть_света	Числовой	внешний ключ (к Часть_света)
	Климат	Числовой	внешний ключ (к Климат)
	Направленность_отдыха	Числовой	внешний ключ (к Страна_направленность)
	Рекомендуемый_сезон	Числовой	внешний ключ (к Рек_сезон)

Название таблицы	Имя поля	Тип данных, длина	Примечание
	Отличие_культуры	Логический	обязательное поле
Город	idCity	Счетчик	первичный ключ
	Название_города	Короткий текст (40)	обязательное поле
	Страна	Числовой	внешний ключ (к Страна)
	Инфраструктура	Числовой	внешний ключ (к Инфраструктура_города)
	Фото	Поле объекта OLE	Изображение города
	Размер города	Числовой	внешний ключ (к Размеры_городов)
	Описание	Длинный текст	Описание города
Инфраструктура_город	id	Счетчик	первичный ключ
	idCity	Числовой	внешний ключ (к Город)
	idR	Числовой	внешний ключ (к Инфраструктура)
Инфраструктура	idR	Счетчик	первичный ключ
	Название_инф	Короткий текст (20)	обязательное поле
Размеры_городов	idSize	Счетчик	первичный ключ
	Размер	Короткий текст (15)	размер города
	Описание	Короткий текст (30)	пояснение размера
Страна_направленность	id	Счетчик	первичный ключ
	idCountry	Числовой	внешний ключ (к Страна)
	idN	Числовой	внешний ключ (к Направленность_отдыха)
Направленность_отдыха	idN	Счетчик	первичный ключ
	Название_направленности	Короткий текст (25)	обязательное поле
Рек_сезон	id	Счетчик	первичный ключ
	idCountry	Числовой	внешний ключ (к Страна)
	idSeason	Числовой	внешний ключ (к Сезоны)

Название таблицы	Имя поля	Тип данных, длина	Примечание
Сезоны	idSeason	Счетчик	первичный ключ
	Сезон	Короткий текст (20)	обязательное поле
Часть_света	idPlace	Счетчик	первичный ключ
	Часть_света	Короткий текст (15)	обязательное поле
Климат	idClimate	Счетчик	первичный ключ
	Тип_климата	Короткий текст (25)	обязательное поле

В итоге схема данных Access примет следующий вид:

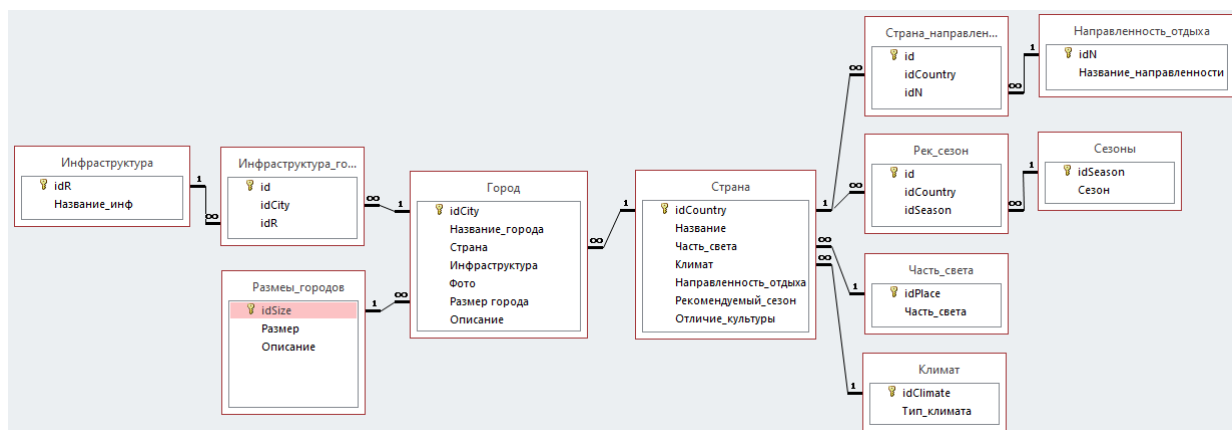


Рисунок 9 – Схема данных Access

## **Разработка программного обеспечения**

Для программной реализации задания была выбрана интегрированная среда разработки Embarcadero RAD Studio, а язык программирования - Delphi

### **3.1 Разработка пользовательских интерфейсов**

В настоящее время каждую программу сопровождают пользовательским интерфейсом, который является неотъемлемой частью любой выполняемой программы.

Пользовательский интерфейс базы данных – это комплекс программ, который организует диалог пользователя с базой данных и реализует сопряжения пользователя с информацией, хранящейся в этой базе данных. Пользовательский интерфейс создает удобную среду для работы с БД.

#### **Формы**

В компонентно-ориентированном программировании форма является представлением окна графического интерфейса пользователя. Форма содержит компоненты и элементы управления. Эти объекты обеспечивают высокоуровневую абстракцию стандартных или настраиваемых виджетов, которыми обычно гораздо проще манипулировать, чем базовым API графического интерфейса. Во время разработки визуальные элементы управления (кнопки, текстовые поля и т. д.) и невидимые компоненты (таймеры, соединения с базой данных, средства компоновки и т. д.) размещаются на форме. Эти элементы управления и компоненты позиционируются и изменяются по размеру в интерактивном режиме, а их свойства и обработчики событий устанавливаются с помощью специального редактора. Во время выполнения автоматически сгенерированный код



создает экземпляры этих элементов управления и компонентов и устанавливает их свойства.

Для реализации визуального интерфейса были выбраны следующие компоненты: Button, ComboBox, Panel, CheckBox, Label, Image, DBImage, DBGrid, DBMemo. Данных компонентов будет достаточно для создания оптимальной среды взаимодействия с программой и представления информации пользователю.

### Схема навигации по формам

Интерфейс программы состоит из трех форм: начальная форма, форма с выборкой городов и форма с добавлением нового города.

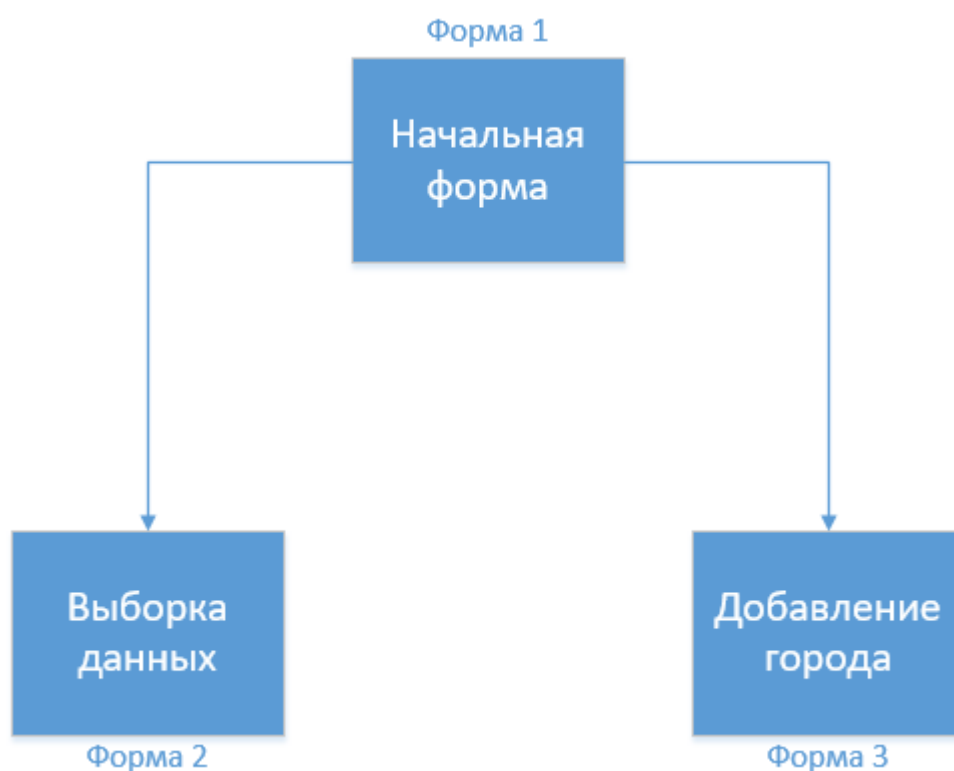


Рисунок 10 – Схема навигации по формам

При запуске программы открывается форма 1. Из нее можно попасть на форму 2 или завершить работу программы. Из формы 2 можно вернуться на форму 1 закрыв ее.

## Компоновка форм

Управляющие элементы и функционально связанные с ними компоненты экрана должны быть зрительно объединены в группы, заголовки которых коротко и четко поясняют их назначение.

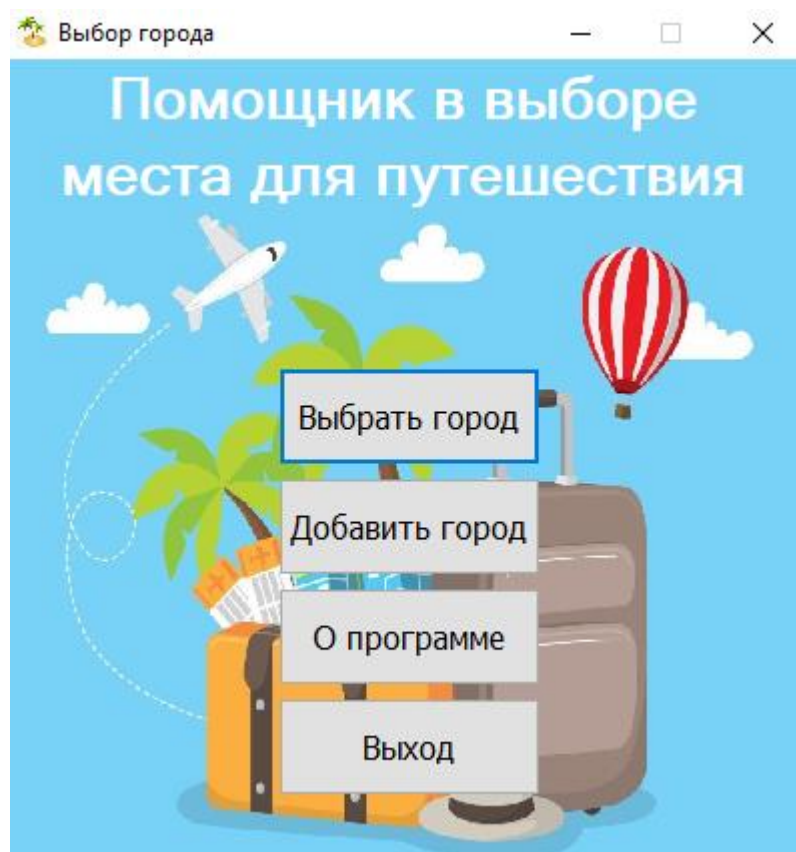
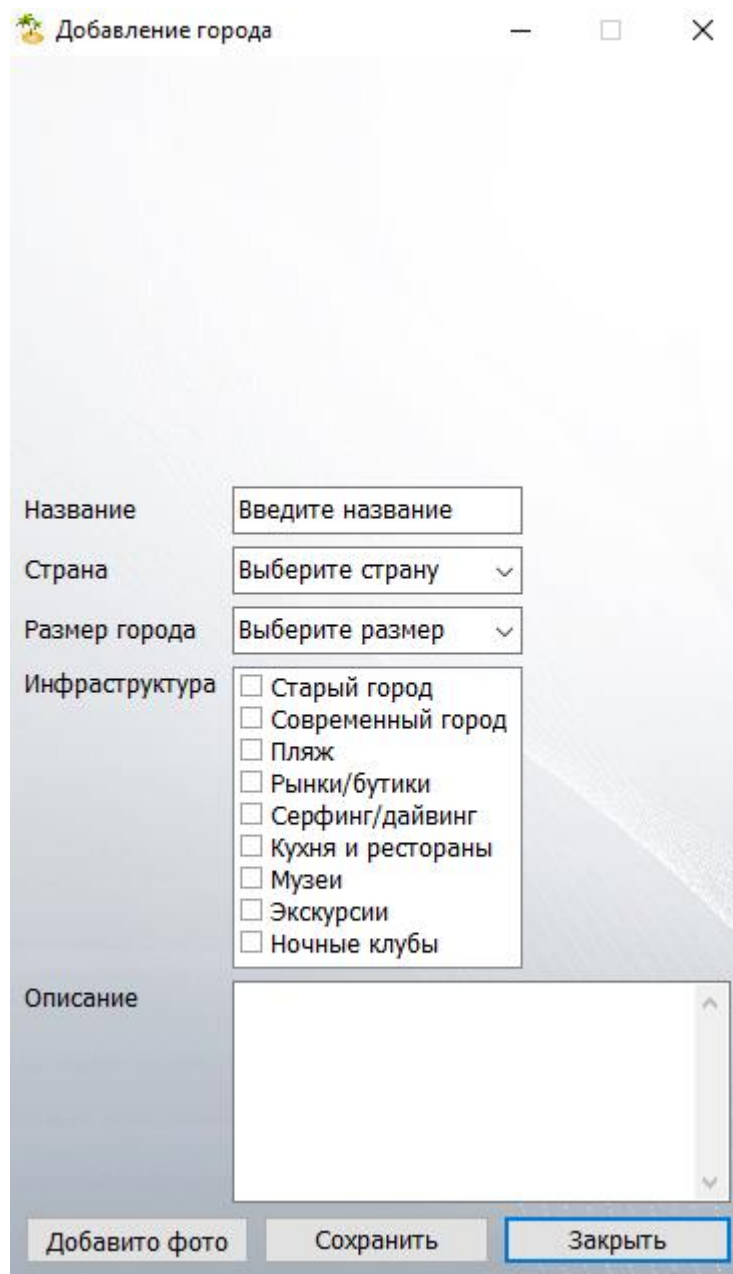


Рисунок 11 – Форма 1 (Главная)





Добавление города

Название: Введите название

Страна: Выберите страну

Размер города: Выберите размер

Инфраструктура:

- ☐ Старый город
- ☐ Современный город
- ☐ Пляж
- ☐ Рынки/бутики
- ☐ Серфинг/дайвинг
- ☐ Кухня и рестораны
- ☐ Музеи
- ☐ Экскурсии
- ☐ Ночные клубы

Описание:

Добавить фото    Сохранить    Заккрыть

Рисунок 13 – Форма 3 (Добавление нового города)

## Эргономика форм

Цель создания эргономичного интерфейса состоит в том, чтобы отобразить информацию настолько эффективно насколько это возможно для человеческого восприятия и структурировать отображение на дисплее таким образом, чтобы привлечь внимание к наиболее важным единицам информации. Основная цель состоит в том, чтобы минимизировать общую

информацию на экране и представить только то, что является необходимым для пользователя.

#### Основные принципы создания эргономичного интерфейса:

1. Естественность (интуитивность). Работа с системой не должна вызывать у пользователя сложностей в поиске необходимых элементов интерфейса для управления процессом решения поставленной задачи.
2. Не избыточность. Пользователь должен вводить только минимальную информацию для работы или управления системой. Например, для получения выборки городов пользователю не обязательно выбирать какие-либо параметры каждого атрибута.
3. Непосредственный доступ к системе помощи. В процессе работы необходимо, чтобы система обеспечивала пользователя необходимыми инструкциями. При открытии формы №2 пользователь получает краткую инструкцию по работе с программой.
4. Гибкость. Насколько хорошо интерфейс системы может обслуживать пользователя с различными уровнями подготовки? Для неопытных пользователей интерфейс может быть организован как иерархическая структура меню, а для опытных пользователей как команды, комбинации нажатий клавиш и параметры. В нашем случае мы выбираем первый вариант.

Применяемый шрифт на всех формах – Tahoma. Это стандартный шрифт Delphi, не имеет засечек, удобен для восприятия. По умолчанию в Delphi размер шрифта – 8 пунктов, но для мониторов с большим разрешением он мал и неудобен для чтения, поэтому выставляем размер 10

пунктов, кроме формы №1, поскольку информации на ней мало и можно увеличить размер шрифта.

### 3.2 Разработка программного обеспечения

#### Подключение базы данных к интерфейсам посредством технологии ADO

Бизнес-логика разрабатываемой программы – взаимодействие с базой данных. В Delphi за подключение БД отвечает невизуальный компонент ADOConnection. После размещения на форме данного компонента нужно настроить его свойство «ConnectionString»: необходимо нажать три точки рядом со свойством > кнопка «Build» > поставщик данных Microsoft Jet 4.0 OLE DB Provider > указать путь к БД. Вместо того, чтобы указать точный путь, можно указать только имя БД, предварительно разместив ее в каталоге с проектом:

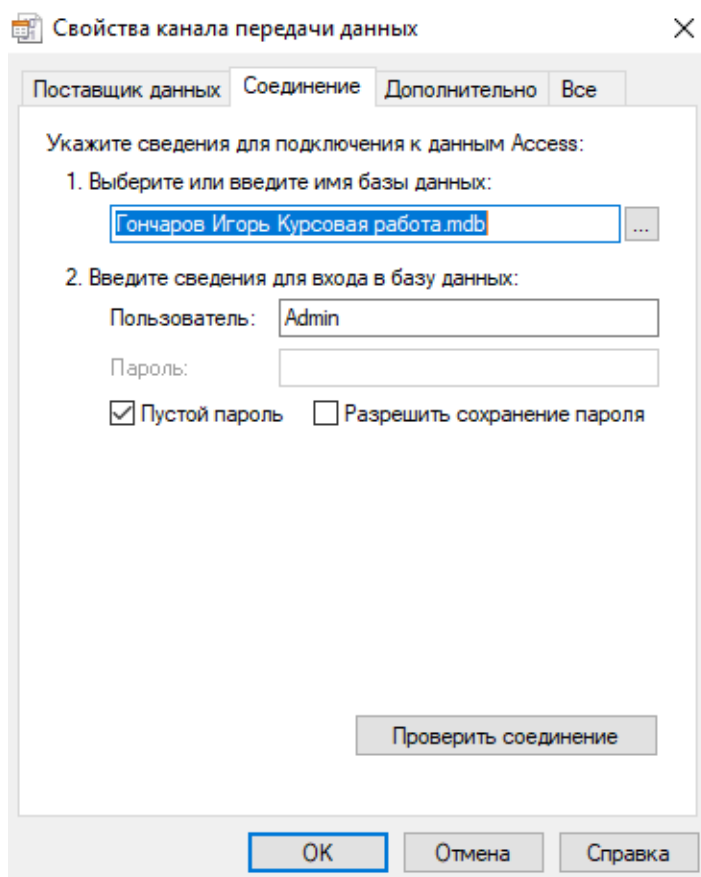


Рисунок 14 – Подключение к БД

В нашем случае логин и пароль устанавливать не требуется, поэтому в этом же окне отметим пункт «Пустой пароль». В свойствах компонента укажем «LoginPrompt» = False. После выполнения указанных действий свойство «ConnectionString» приобретает следующий вид:

```
«Provider=Microsoft.Jet.OLEDB.4.0;User ID=Admin;Data Source=Гончаров  
Игорь Курсовая работа.mdb;Mode=Share Deny None;Jet OLEDB:System  
database="";Jet OLEDB:Registry Path="";Jet OLEDB:Database Password="";Jet  
OLEDB:Engine Type=5;Jet OLEDB:Database Locking Mode=1;Jet  
OLEDB:Global Partial Bulk Ops=2;Jet OLEDB:Global Bulk Transactions=1;Jet  
OLEDB:New Database Password="";Jet OLEDB:Create System  
Database=False;Jet OLEDB:Encrypt Database=False;Jet OLEDB:Don't Copy  
Locale on Compact=False;Jet OLEDB:Compact Without Replica Repair=False;Jet  
OLEDB:SFP=False;»
```

Устанавливаем ADOConnection свойство «Connected» = True. БД подключена к программе.

### **Организация и визуализация выборок данных**

Для организации выборки данных используем невидимый компонент ADOQuery. Подключаем его к ADOConnection выбрав в свойстве «Connection» добавленный ADOConnection. ADOQuery имеет свойство «SQL» - в нем будет находится SQL запрос, который мы динамически будем вносить во время работы программы, и свойство «Active», переключив значение которого на True мы активируем данный компонент.

Невидимый компонент DataSource в Delphi представляет собой источник данных, который обеспечивает связь между набором данных и компонентами отображения и редактирования данных. Основное свойство источника данных – «DataSet». В нем выберем добавленный ADOQuery.

Для визуализации выборки используем визуальный компонент DBGrid. Его главный компонент – «DataSource». В нем укажем добавленный на предыдущем шаге DataSource.

После выполнения этих действий мы уже можем увидеть в DBGrid выборку из БД, соответствующую SQL запросу, находящемуся в ADOQuery:

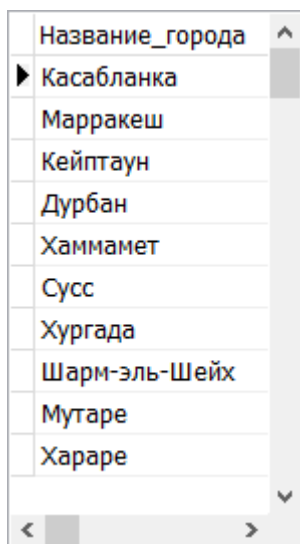


Рисунок 15 – Пример выборки при запросе  
"select Название\_города from Город"

### Поиск информации в базе данных

Основной задачей программы является получение выборки городов и стран. Получать ее мы будем по нажатию Button на форме. После размещения «Button1» назначим ей свойство «Caption» = «Показать города» зайдя в ее события и выберем событие OnClick. Созданная процедура будет задавать SQL запрос для ADOQuery. Внутри процедуры размещаем следующий код:



```

procedure TForm3.Button1Click(Sender: TObject);
begin
    adoquery1.Close;
    adoquery1.sql.Clear;
    adoquery1.sql.add('SELECT distinct Город.Название_города, (
    adoquery1.sql.add('FROM (((Часть_света INNER JOIN ((Климат
    adoquery1.sql.add('Часть_света.[idPlace] = Страна.[Часть_с
    adoquery1.sql.add('Город.[Страна]) INNER JOIN (Инфраструкт
    adoquery1.open;
end;

```

Рисунок 16 – Вставка SQL запроса

SQL запрос слишком длинный, поэтому его приходится разбивать на несколько строк. Полный текст запроса:

«SELECT distinct Город.Название\_города, Страна.Название, Город.Описание  
 FROM (((Часть\_света INNER JOIN ((Климат INNER JOIN Страна ON  
 Климат.[idClimate] = Страна.[Климат]) INNER JOIN  
 (Направленность\_отдыха INNER JOIN Страна\_направленность ON  
 Направленность\_отдыха.[idN] = Страна\_направленность.[idN]) ON  
 Страна.[idCountry] = Страна\_направленность.[idCountry]) ON  
 Часть\_света.[idPlace] = Страна.[Часть\_света]) INNER JOIN (Сезоны INNER  
 JOIN Рек\_сезон ON Сезоны.[idSeason] = Рек\_сезон.[idSeason]) ON  
 Страна.[idCountry] = Рек\_сезон.[idCountry]) INNER JOIN (Размеры\_городов  
 INNER JOIN Город ON Размеры\_городов.[idSize] = Город.[Размер города])  
 ON Страна.[idCountry] =Город.[Страна]) INNER JOIN (Инфраструктура  
 INNER JOIN Инфраструктура\_город ON Инфраструктура.[idR] =  
 Инфраструктура\_город.[idR]) ON Город.[idCity] =  
 Инфраструктура\_город.[idCity]».

После нажатия на «Button1» в свойство «SQL» у «ADOQuery1» будет добавлен данный SQL запрос, и мы получим выборку городов, их описаний и стран, в которых они находятся (выборка отображается в DBGrid).

Отбор нужных пользователю городов мы будем осуществлять при помощи ComboBox и CheckBox. Выборка в ADOQuery должна меняться в зависимости от нажатых CheckBox и текста в ComboBox. Для этого нам нужно задать параметры в SQL запросе – сколько значений у атрибута – столько и параметров. Преобразуем запрос, представленный выше, добавив в него следующие параметры:

```
«where (Часть_света.Часть_света=:Place1 or Часть_света.Часть_света=:Place2  
or Часть_света.Часть_света=:Place3 or Часть_света.Часть_света=:Place4 or  
Часть_света.Часть_света=:Place5) and (Сезоны.Сезон=:Season) and  
(Тип_климата=:Climate1 or Тип_климата=:Climate2 or  
Тип_климата=:Climate3 or Тип_климата=:Climate4 or Тип_климата=:Climate5  
or Тип_климата=:Climate6 or Тип_климата=:Climate7 or  
Тип_климата=:Climate8 or Тип_климата=:Climate9) and  
(Направленность_отдыха.Название_направленности=:Napra1 or  
Направленность_отдыха.Название_направленности=:Napra2 or  
Направленность_отдыха.Название_направленности=:Napra3 or  
Направленность_отдыха.Название_направленности=:Napra4) and  
(Страна.Отличие_культуры=:Culture1 or  
Страна.Отличие_культуры=:Culture2) and (Размеры_городов.Размер=:Size1 or  
Размеры_городов.Размер=:Size2 or Размеры_городов.Размер=:Size3 or  
Размеры_городов.Размер=:Size4) and (Название_инф=:Inf1 or  
Название_инф=:Inf2 or Название_инф=:Inf3 or Название_инф=:Inf4 or  
Название_инф=:Inf5 or Название_инф=:Inf6 or Название_инф=:Inf7 or  
Название_инф=:Inf8 or Название_инф=:Inf9)».
```

На форме 2 для параметра «Сезон» создадим один ComboBox, для остальных параметров по одному CheckBox на каждое возможное значение (всего 33 компонента).

Свойство «Caption» у CheckBox отвечает за видимый текст компонента. Установим для каждого CheckBox свойство «Caption» соответствующее какому-либо значению параметра:

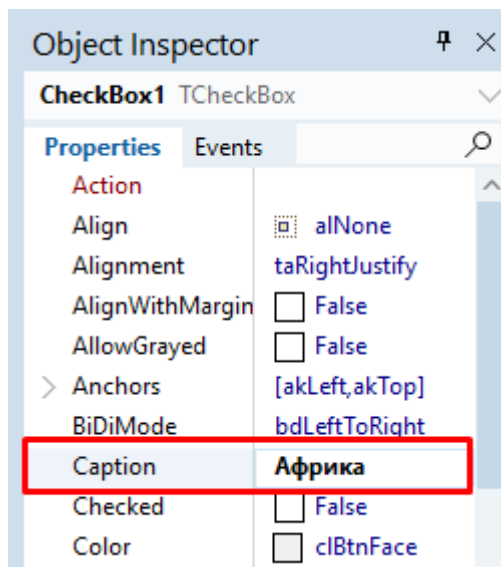


Рисунок 17 – Установка «Caption»

Теперь необходимо проверить состояния каждого CheckBox – если он выбран (свойство «Checked» = True), то значение соответствующего ему параметра принимает значения свойства «Caption» этого CheckBox, в ином случае значение параметра становится пустым (вставляется пробел):

```
if checkbox1.Checked=true then  
ADOQuery1.Parameters.ParamByName('Placel').Value:=checkbox1.caption else  
ADOQuery1.Parameters.ParamByName('Placel').Value:='';
```

Рисунок 18 – проверка CheckBox

Этот же код повторим для других параметров кроме «Сезон». Его будем выбирать из ComboBox – параметр «Season» всегда будет принимать значение свойства «Text» у ComboBox:

```
ADOQuery1.Parameters.ParamByName('Season').Value:=combobox1.Text;
```

Рисунок 19 – Параметр «Сезон» из ComboBox

Теперь после установления галочки на CheckBox и выбора из ComboBox, по нажатию «Button1» пользователь увидит в DBGrid соответствующую выборку из БД.

Если пользователь не выбрал ни один CheckBox – выборка будет пуста. Возможное решение: проверяем все CheckBox, относящиеся к одной группе параметров, если ни один из них не выбран, то значение каждого параметра из этой группы становится равно «Caption» одного из CheckBox:

```
if ((checkbox1.Checked=false) and (checkbox2.Checked=false) and  
(checkbox3.Checked=false) and (checkbox4.Checked=false) and (checkbox5.Checked=false)) then  
begin  
ADOQuery1.Parameters.ParamByName('Place1').Value:=checkbox1.caption;  
ADOQuery1.Parameters.ParamByName('Place2').Value:=checkbox2.caption;  
ADOQuery1.Parameters.ParamByName('Place3').Value:=checkbox3.caption;  
ADOQuery1.Parameters.ParamByName('Place4').Value:=checkbox4.caption;  
ADOQuery1.Parameters.ParamByName('Place5').Value:=checkbox5.caption;  
end;
```

Рисунок 20 – Избавляемся от пустой выборки

В таком случае в выборку будут включены все города.

Решение для аналогичной ситуации с ComboBox (не выбрано значение):

```
if (combobox1.text='Выберите подходящий сезон') then  
showmessage('Выберите сезон!');
```

Рисунок 21 – Пустая выборка ComboBox

Пользователю будет показано сообщение с предложением выбрать сезон.

Если после выбора всех необходимых параметров выборка осталась пуста (по причине отсутствия в БД подходящих городов), предложим пользователю изменить параметры поиска:

```
if (DBGrid1.DataSource.DataSet.IsEmpty) and  
NOT(combobox1.text='Выберите подходящий сезон') then begin  
showmessage('Ничего не найдено!'+#13#10+'Измените параметры поиска и повторите запрос еще раз.');
```

Рисунок 22 – Предложение изменить параметры поиска

Это сообщение появится только если выбрано какое-либо значение в ComboBox.

На форме 2 для атрибутов «Тип климата», «Размер города», «Отличие в культуре» добавим подсказки: рядом с их компонентом Label добавим Button, по нажатию на который, пользователь увидит сообщение с описанием атрибута:

```
procedure TForm3.Button3Click(Sender: TObject);  
begin  
showmessage('Климат является универсальным лечебным фактором на курортах, где климатические факторы определяемые высотой над уровнем моря, широтой и долготой.');
```

```
procedure TForm3.Button5Click(Sender: TObject);  
begin  
showmessage('Отличие от европейской культуры. Например, если, приехав из Европы, вы увидите, что в России принято ходить в сапогах, то это не так. В России принято ходить в обуви, которая не имеет подошвы, а имеет шпильку.');
```

```
procedure TForm3.Button6Click(Sender: TObject);  
begin  
showmessage('Размер города Описание'+#13#10+  
          'Малый до 20 тыс. жителей'+#13#10+  
          'Средний 20 - 100 тыс. жителей'+#13#10+  
          'Крупный 100-500 тыс. жителей'+#13#10+  
          'Крупнейший более 500 тыс. жителей');
```

Рисунок 23 – Подсказки

## Управление выборками по принципу «хозяин – подчиненный»

В SQL запросе мы используем оператор DISTINCT. Он используется для удаления дубликатов из результирующего набора оператора SELECT. Дубликаты городов в выборке возникают из-за наличия в БД связующих таблиц «Инфраструктура\_город» и «Страна\_направленность». Оператор DISTINCT решает эту проблему, но, в то же время накладывает ограничения: в выборке длина всех полей ограничивается 255 символами и нельзя использовать «Поле объекта OLE». Это значит, что описание будет урезано, а выборка фотографий города совсем невозможна.

Необходимо создать дополнительные ADOQuery и DataSource и организовать следующую логику: при выборе какого-либо города в DBGrid, выборка в «ADOQuery2» должна меняться таким образом, чтобы в нее попадал тот город, который выбран в данный момент. В это же время в DBMemo и DBImage появляются соответствующие описание и изображение города. Для этого добавим событие «DBGrid1CellClick». В этом событии запишем в свойство «SQL» у «ADOQuery2» выборку всех атрибутов таблицы «Город», где атрибут «Название\_города» будет некоторым параметром. Далее задаем значение параметра равным значению этого же параметра из «ADOQuery1»:

```
procedure TForm3.DBGrid1CellClick(Column: TColumn);
begin
  adoquery2.Close;
  adoquery2.sql.Text:='select * from Город where (Название_города=:n)';
  ADOQuery2.Parameters.ParamByName('n').value:=ADOQuery1['Название_города'];
  adoquery2.open;
  dbmemo1.DataField:='Описание';
  dbimage1.DataField:='Фото';
end;
```

Рисунок 24 – «Хозяин-подчиненный»

Для вывода изображения и описания задаем свойства «DataField» у «DBMemo» = «Описание» и «DBImage» = «Фото», предварительно

подключив их к «DataSource2». Добавим этот же код к событию «DBGrid1DrawColumnCell» у DBGrid, чтобы при его прокрутке описание и изображения менялись.

## Добавление информации в БД

Добавление нового города осуществляется на форме 3. На ней размещены необходимые компоненты для записи в них атрибутов города (рисунок 13). После ввода значений и нажатия на кнопку «Сохранить», информация будет занесена в БД.

Создаем процедуру SaveCityName. Внутри этой процедуры происходит запись названия города, его страны, размера города и его описание:

```
procedure TForm4.SaveCityName(gname:string);
var id:integer;
j: tjpegimage;
b: tbitmap;
aqt: tadoquery;
begin
    adoquery3.open;
    adoquery3.insert;
    adoquery3['Название_города']:=edit1.Text;

    aqt:=tadoquery.Create(form4);
    aqt.Connection:=form3.ADOConnection1;
    aqt.SQL.Text:='select idCountry from Страна where Название=:n';
    aqt.Parameters.ParamByName('n').value:=combobox1.Text;
    aqt.Open;
    adoquery3['Страна']:=aqt.FieldByName('idCountry').Value;
    aqt.Free;

    if combobox2.Text='Малый' then adoquery3['Размер города']:= '1';
    if combobox2.Text='Средний' then adoquery3['Размер города']:= '2';
    if combobox2.Text='Крупный' then adoquery3['Размер города']:= '3';
    if combobox2.Text='Крупнейший' then adoquery3['Размер города']:= '4';

    adoquery3['Описание']:=memol1.text;
    id:=adoquery6.FieldByName('Expr1000').Value;
    id:=id+1;
    adoquery3['Инфраструктура']:=id;

    adoquery3.post;
end;
```

Рисунок 25 – Создание процедуры сохранения города

В разделе public создаем две функции:

```
public

Function GetIDCity(gname:string):integer;
Function GetIDInfr(gname:string):integer;
```

Рисунок 26 – Объявление функций

С их помощью получим нужный id записи города:

```
function TForm4.GetIDCity(gname: string): integer;
var aq:tadoquery;
begin
    aq:=tadoquery.Create(form4);
    aq.Connection:=form3.ADOConnection1;
    aq.SQL.Text:='select idCity from Город where Название_города=:n';
    aq.Parameters.ParamByName('n').Value:=gName;
    aq.Open;
    if aq.RecordCount>0 then result:=aq.Fields[0].AsInteger else result:=-1;
    aq.Free;
end;

function TForm4.GetIDInfr(gname: string): integer;
var aq:tadoquery;
begin
    aq:=tadoquery.Create(form4);
    aq.Connection:=form3.ADOConnection1;
    aq.SQL.Text:='select idR from Инфраструктура where Название_инф=:n';
    aq.Parameters.ParamByName('n').Value:=gName;
    aq.Open;
    if aq.RecordCount>0 then result:=aq.Fields[0].AsInteger else result:=-1;
    aq.Free;
end;
```

Рисунок 27 – Описание функций



Весь обработчик события «OnClick» кнопки «Сохранить» примет следующий вид:

```
procedure TForm4.Button2Click(Sender: TObject);
var i:integer;
begin
adoquery6.close;
adoquery6.open;
SaveCityName(edit1.text);
for i := 0 to checklistbox1.Items.Count-1 do
if checklistbox1.Checked[i] then
begin
adoquery5.open;
adoquery5.Insert;
adoquery5['idCity']:=GetIDCity(Edit1.text);
adoquery5['idR']:=GetIDInfr(CheckListBox1.items[i]);
adoquery5.post;
end;
end;
```

Рисунок 28 – Обработчик кнопки «Сохранить»

Заполнение ComboBox названиями стран можно реализовать так (в adoquery7 содержится выборка названий всех стран):

```
procedure TForm4.FormActivate(Sender: TObject);
begin
ADOquery7.Open;

with ADOquery7 do
begin
ComboBox1.Items.clear;
DisableControls;
ComboBox1.Items.BeginUpdate;
First;
while not Eof do
begin
ComboBox1.Items.Add(FieldByName('Название').AsString);
Next;
end;
ComboBox1.Items.EndUpdate;
EnableControls;
end;
end;
```

Рисунок 29 – Заполнение ComboBox

Поскольку на этапе создания схемы данных в Access мы включили обеспечение целостности, вводится будет только корректная информация. Дополнительным ограничением случит тот факт, что название страны, размер города и инфраструктуру можно выбрать только из готовых наборов атрибутов, не внося ничего нового (чего нет в БД). Пользователю остается только вписать название и описание города, а также его описание.

### **Хранение в базах данных Access графической информации**

Для отображения изображений из БД в программе необходимо, чтобы в Access, в «Поле объекта OLE», где хранится изображение, появилась надпись: «Двоичные данные». Добиться этого можно, например, загружая изображение непосредственно из программы.

Размещаем на форме 3 невидимый компонент Opendialog. При нажатии на Button «Добавить фото» открывается «Проводник», в котором выбирается необходимое изображение (в формате jpeg или bmp):

```
procedure TForm4.BitBtn1Click(Sender: TObject);  
begin  
  if Opendialog1.execute then  
  begin  
    image1.picture.loadfromfile(opendialog1.filename);  
  end;  
end;
```

Рисунок 30 – Загрузка файла в OpenFileDialog

При нажатии кнопки «Сохранить» происходит проверка загруженного файла: если он имеет формат bmp, то будет вставлен в поле «Фото», в ином случае выполнится его преобразование в формат bmp. Происходит это путем создания двух переменных типа «tjregimage» и «tbitmap». В переменную «tjregimage» загружается изображение, и данная переменная присваивается

переменной «tbitmap», после чего происходит вставка переменной «tbitmap» в выбранное поле выборки (в DBGrid), а переменные освобождаются:

```
procedure TForm4.BitBtn2Click(Sender: TObject);
var
j: tjpegimage;
b: tbitmap;
begin
    adoquery2.edit;
    if extractfileext(opendialog1.filename)='.bmp' then
        TBlobField(adoquery2.fieldbyname('foto')).assign(imagel.picture)
    else
        begin
            j:=tjpegimage.create;
            try
                j.compressionquality:=100;
                j.loadfromfile(opendialog1.filename);
                b:=tbitmap.create;
                try
                    b.assign(j);
                    tblobfield(adoquery2.fieldbyname('foto')).assign(b);
                finally
                    b.free;
                end;
            finally
                j.Free;
            end;
        end;
    adoquery2.Post;
end;
```

Рисунок 31 – Загрузка изображения в выборку и его сохранение

### 3.3 Тестирование информационной системы

Тестирование ИС – это процесс выполнения ПО системы или компонента в условиях анализа или записи получаемых результатов с целью проверки некоторых свойств тестируемого объекта. Тестирование позволяет выявить подавляющее большинство ошибок, допущенных при составлении программ.

Тестирование проводилось путем выполнения ПО в его окончательной конфигурации, интегрированной с другими программными и аппаратными

системами (БД Access). Был осуществлен запуск всех возможных сценариев работы с ПО. Ниже схематично изображен процесс тестирования и его результаты.

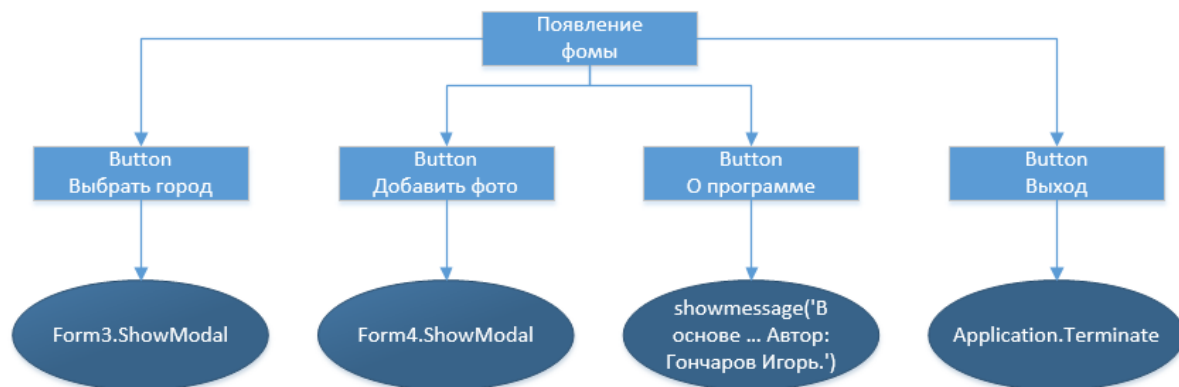


Рисунок 32 – Сценарий работы с формой 1

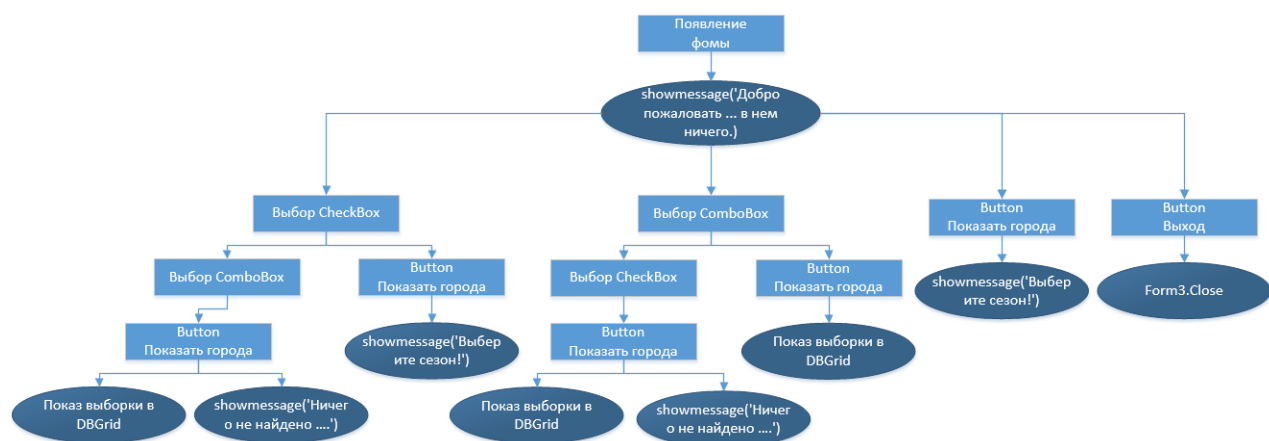


Рисунок 33 – Сценарий работы с формой 2

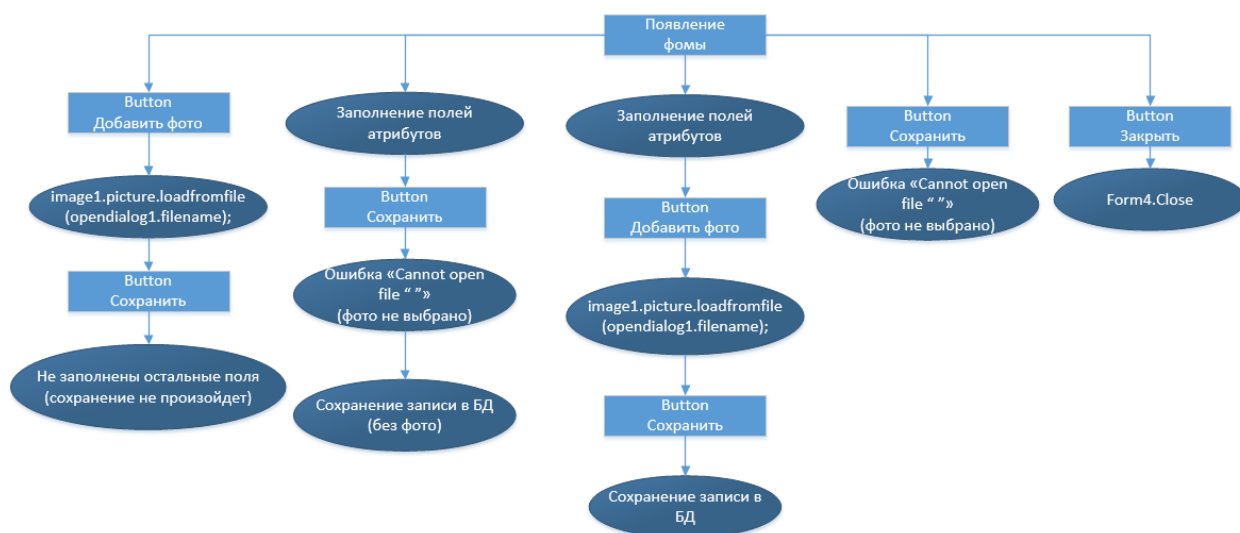


Рисунок 34 – Сценарий работы с формой 3

### Результаты тестирования

1. При выборе большого количества ComboBox на форме 2, выборка иногда бывает пуста по причине отсутствия в БД городов с заданными параметрами. Возможное решение – добавить в БД больше городов или упразднить некоторые атрибуты.
2. Если на форме 3 нажать кнопку сохранить, при этом не выбрав фото, произойдет сохранение записи с БД без фото. Возможное решение – добавить проверку на наличие фото в opendialog.

## **Заключение**

Результатом выполнения курсовой работы являются база данных и прикладная программа для нее, служащие информационной системой, которая решает задачи выбора определенной страны и города в зависимости от предпочтений пользователя.

Разработанная ИС отвечает всем требованиям выделенной предметной области, таблицы созданной БД отвечают требованиям нормализации, что обеспечивает целостность и непротиворечивость информации. Средствами Delphi создан удобный GUI.

На этапе тестирования были выявлены легкоустраняемые недостатки, не влияющие на работоспособность программы.

Разработанная ИС позволяет решать все задачи, сформулированные в целях курсовой работы.

В качестве перспектив развития ИС можно назвать создание аналогичного сайта с возможностью добавления галереи с фотографиями города и с расширенным описанием его конкретных достопримечательностей.

## Список источников

1. Голицына О. Л., Партыка Т. Л., Попов И. И. Основы проектирования баз данных; Форум - Москва, 2012. - 416 с.
2. Гринченко, Н.Н. и др. Проектирование баз данных. СУБД Microsoft Access; Горячая Линия Телеком - Москва, 2011. - 240 с.
3. Дейт, К.Дж. Введение в системы баз данных; К.: Диалектика; Издание 6-е - Москва, 2010. - 784 с.
4. Теория и практика построения баз данных. 8-е изд./Д. Крёнке. – СПб.: Питер, 2005.- 800с.
5. Малыхина М.П. Базы данных: основы, проектирование, использование учеб. пособие / М.П.Малыхина. – СПб. БХВ-Петербург, 2004. – 512с.
6. Фаронов В.В., Шумаков П.В. DELPHI Руководство разработчика баз данных, М: Нолидж, 2007. – 640с.
7. Кириллов В.В. Основы проектирования реляционных баз данных [Электронный ресурс] – Режим доступа - <http://www.cit-forum.com>.
8. Кузнецов С. Д. Основы современных баз данных [Электронный ресурс] – Режим доступа - <http://www.cit-forum.com>.
9. Д. Осипов. Базы данных и Delphi. Теория и практика. - СПб.: БХВ-Петербург, 2011. – 752 с.
10. Илюшечкин В.М. Основы использования и проектирования баз данных: учебник для СПО. Издательство Юрайт, 2019. – 213с.

## Тезаурус

БД – база данных

СУБД – система управления базами данных

GUI – графический интерфейс пользователя

SQL – язык, осуществляющий запросы в БД посредством СУБД

Кортеж – строка в отношении

ИС – информационная система