# АННОТАЦИЯ

В данной курсовой работе разработана информационная система «Туристическое агентство».

Целью работы является разработка информационной системы в формате веб-приложения для предметной области «Туристическое агентство». Данная система удовлетворяет потребности большинства компаний на рынке в сфере туризма.

В результате проведённых работ по реализации информационной системы разработана программа для «Туристического агентства» с удобным пользовательским интерфейсом и широким функционалом.

Содержание

[АННОТАЦИЯ 1](#_Toc12838290)

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ 3](#_Toc12838291)

[ВВЕДЕНИЕ 4](#_Toc12838292)

[1. ПОСТАНОВКА ЗАДАЧИ 5](#_Toc12838293)

[2. ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ 6](#_Toc12838294)

[3. ИНФОРМАЦИОННОЕ МОДЕЛИРОВАНИЕ 8](#_Toc12838295)

[4. ВЫБОР ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 12](#_Toc12838296)

[5. СОЗДАНИЕ БАЗЫ ДАННЫХ 13](#_Toc12838297)

[6. РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БАЗЕ ДАННЫХ 15](#_Toc12838298)

[7. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 18](#_Toc12838299)

[8. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 21](#_Toc12838300)

[ЗАКЛЮЧЕНИЕ 23](#_Toc12838301)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 24](#_Toc12838302)

[ПРИЛОЖЕНИЕ А 25](#_Toc12838303)

[ПРИЛОЖЕНИЕ Б 31](#_Toc12838304)

[ПРИЛОЖЕНИЕ В 36](#_Toc12838305)

[ПРИЛОЖЕНИЕ Г 46](#_Toc12838306)

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ

MVP – Model-View-Presenter

MVC – Model-View-Controller

UI – User Interface

ПрО – предметная область

ИС – информационная система.

# ВВЕДЕНИЕ

Украинские туристические предприятия, вступающие на международный рынок, сталкиваются в своей деятельности с проблемами освоения новых информационных технологий, которые являются необходимым условием международной интеграции и современной концепции туристического бизнеса как информационно-насыщенной сферы. Именно поэтому особенно актуальным представляется создание такого инструмента, который бы помог минимизировать затраты времени и рабочих ресурсов, облегчить работу таких компаний .

Целью работы является разработка информационной системы в формате веб-приложения для предметной области «Туристическое агентство». Оно поможет туристу самостоятельно выбрать тур, соответствующий его критериям, а также сэкономить время на регистрацию новых клиентов и работников агентства, проводить анализ статистических данных для администратора приложения. Любой звонок, визит, пожелание клиента - решается в кратчайшие сроки.

Для достижения указанной цели в курсовой работе необходимо решить следующие задачи:

1. анализ предметной области;
2. формирование требований к создаваемой программе;
3. проектирование функциональной модели;
4. разработка архитектуры;
5. проектирование БД ;
6. выбор технологий и средств реализации программы;
7. реализация клиентского приложения, которое даст возможность удобно манипулировать данными ПрО для различных категорий пользователей;
8. обеспечение разграничения прав доступа к приложению со стороны различных категорий пользователей и защиты от несанкционированного доступа.

# ПОСТАНОВКА ЗАДАЧИ

Анализ информационной системы и проектирование базы данных на основе категорий пользователей и бизнес задач, в решении которых им должно помочь данное приложение в данной ПрО. В результате было выделено три категории потенциальных пользователей этим приложением:

1. турист – имеет право на просмотр всей информации о турах, поиск туров по заданным критериям, а также может зарегистрироваться самостоятельно на определённый тур;
2. сотрудник – имеет права на просмотр всей информации о клиентах и её редактирование, регистрацию новых клиентов, а также имеет право на просмотр информации о турах;
3. администратор – может регистрировать новые туры, редактировать всё, что связанно с туром, имеет право на добавление, редактирование и удаление сотрудников, на просмотр правильности заполнения данных о туре, выдачи премии сотрудникам;

Таким образом, каждый пользователь данной информационной системы имеет свою роль, а основываясь на этой роли, у него есть набор доступных функций. Такая информационная система позволяет решить следующие проблемы:

1. для любителей отдыха данная программа систематизирует и упрощает процесс оформления тура;
2. для сотрудников программа упрощает процесс регистрации клиента на определенный тур.
3. для администратора программа упрощает процесс добавления новых сотрудников, туров, а также редактирования данных в соответствующих таблицах, а также анализ популярности посещения тех или иных стран, городов;

Полный список задач, записанных в формате пользовательских историй [1], можно просмотреть в Приложении А.

# ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Для реализации информационной системы выбрана трехуровневая архитектура клиент-сервер. В этой архитектуре между клиентом и сервером БД существует еще один уровень – сервер приложений, который для сервера БД является клиентом, а для клиента – сервером. На сервере приложений сохраняется программное обеспечение, которое содержит основную часть бизнес-логики данной ИС (основные алгоритмы обработки данных, полученных от сервера БД). Сервер приложений анализирует, что именно нужно пользователю и посылает серверу БД соответствующий запрос. Сервер БД в свою очередь обеспечивает не только сохранение данных, но и их целостность. Именно он выполняет запросы сервера приложений и обеспечивает разграничение прав доступа пользователей.

Трехуровневая архитектура обладает рядом преимуществ:

1. модель может быть расширена до многоуровневой, что позволяет повысить гибкость и масштабируемость создаваемых приложений.
2. изолированность уровней друг от друга позволяет быстро и просто переконфигурировать систему при возникновении сбоев или при плановом обслуживании на одном из уровней;

Так как используется трехуровневая архитектура, будем использовать паттерн MVC. Этот шаблон разделяет работу веб-приложения на три отдельные функциональные роли: модель данных (model), пользовательский интерфейс (view) и управляющую логику (controller). Таким образом, изменения, вносимые в один из компонентов, оказывают минимально возможное воздействие на другие компоненты.

В данном паттерне модель не зависит от представления или управляющей логики, что делает возможным проектирование модели как независимого компонента и, например, создавать несколько представлений для одной модели.

Впервые этот шаблон был применён в фреймворке, разрабатываемом для языка Smalltalk в конце 1970-х годов. С этого момента он играет основополагающую роль в большинстве фреймфорков с пользовательским интерфейсом. Он в корне изменил взгляд на проектирование приложений.

Большинство фреймворков для веб-программирования сейчас в основе своей содержат именно MVC. К наиболее удачным примерам применения этого паттерна для языка Python можно отнести Django.



Рисунок 1.1 – Диаграмма работы паттерна MVС

# ИНФОРМАЦИОННОЕ МОДЕЛИРОВАНИЕ

Первый этап проектирования заключается в создании схемы БД, которая включает определение сущностей и существующих между ними связями и атрибутами. Результатом данного проектирования является схема БД, представленная в виде ER-диаграммы (Entity-Relationship), на которой отображаются основные сущности ПрО, их атрибуты и связи. Схема БД создаётся на основе функциональных требований пользователей и абсолютно не зависит от каких-либо особенностей физической реализации информационной системы, таких как тип выбранной СУБД или язык программирования.

Последовательность этапов создания схемы базы данных:

1. определение сущностей;
2. определение атрибутов сущностей;
3. определение связей между сущностями;
4. задание первичных и альтернативных ключей.

В данной курсовой работе поставлена задача построения ER-диаграммы для туристического агентства. Исходя из анализа ПрО, для функционирования информационной системы необходимы следующие сущности, которые имеют первичный ключ и атрибуты:

1. Страна: id – первичный ключ, название страны;
2. Город: id – первичный ключ, название города;
3. Клиент: id – первичный ключ, ФИО, номер телефона, номер и серия паспорта;
4. Сотрудник: id – первичный ключ, ФИО, зарплата, премия;
5. Место проживания: id – первичный ключ, тип, условия номера;
6. Транспорт: id – первичный ключ, тип, количество мест;
7. Тур: id – первичный ключ, цена, описание, дата окончания действия, длительность, максимальное количество туристов.
8. Страховочное агентство: id – первичный ключ, вид страховки, название компании, сумма выплаты.
9. Отзывы о сотрудниках: id – первичный ключ, id сотрудника, id клиента, текст отзыва, оценка (от 1 до 5 ).
10. Отзывы о турах: id – первичный ключ, id клиента, id тура, текст отзыва, оценка

(от 1 до 5 ).

1. Отчет: id – первичный ключ, id сотрудника, id регистрации.
2. Запрос на регистрацию: id – первичный ключ, id клиента, id тура .
3. Запрос на обратный звонок: id – первичный ключ, id клиента, номер телефона.

В данной ПрО можно выделить следующие связи между сущностями:

1. так как клиент может оформить много туров, а у тура может быть много клиентов, то между ними образовывается связь «многие-ко-многим»;
2. так как у тура может быть много мест проживаний, а у мест проживания может быть много туров, то между ними образовывается связь «многие-ко-многим»;
3. так как транспорт может перевозить клиентов в разные туры, а у тура может быть много транспортов, то между ними образовывается связь «многие-ко-многим»;
4. так как страховочное агентство может обслуживать разные туры, а у тура может быть несколько страховок, то между ними образовывается связь «многие-ко-многим»;
5. так как за конкретный тур могут отвечать несколько сотрудников, а за несколько туров может быть ответственен один сотрудник, то между ними образовывается связь «многие-ко-многим»;
6. так как тур может проходить во многих городах, а у одного города может быть много туров, то между ними образовывается связь «многие-ко-многим»;
7. так как тур может проходить через много стран, а у каждой страны может быть много туров, то между ними образовывается связь «многие-ко-многим».
8. так как у тура могут быть несколько туристов, а один клиент может быть зарегистрирован на несколько туров, то между ними образовывается связь «многие-ко-многим»;
9. так как у сотрудника может быть один отчёт по одному клиенту по одному туру, то между ними образовывается связь «один-ко-одному»;
10. так как у клиента может быть один отзыв по одному туру, то между ними образовывается связь «один-ко-одному»;
11. так как у клиента может быть один отзыв по одному сотруднику, то между ними образовывается связь «один-ко-одному»;

Следующим шагом является формализация связей между сущностями. Так как между Туром, Клиентом и Сотрудником образовалась n-арная связь, а в реляционных моделях эту связь нельзя представить в виде трёх таблиц, то стоит провести формализацию. Для формализации n-арной связи необходимо n+1 отношение: по одному для каждой сущности, принимающей участие в связи, и одно для самой связи. Причём отношение для связи, точнее, его схема должна включать первичные ключи всех других n отношений. Более того, множество всех этих ключей будет первичным ключом отношения связи. Поэтому была создана таблица «Регистрация», в которой в качестве составного первичного ключа будут выступать первичные ключи сущностей Клиента, Сотрудника и Тура. Каждая часть составного ключа будет ссылаться на другой объект. Все связи этой таблицы будут связями типа «один-ко-многим». Та же операция происходит со связью между сущностями Тур, Город и Страна; и между сущностями Клиент, Тур и Страховочное агентство. Как итог, мы получаем новые сущности «ТГС» и «Страховка». Та же операция происходит со связью между сущностями ТГС, Клиент и Место проживания; и между сущностями Клиент, ТГС и Транспорт. Как итог, мы получаем новые сущности «Проживание» и «Проезд».

После формализации была получена следующая схема, представленная на рис 2.1.

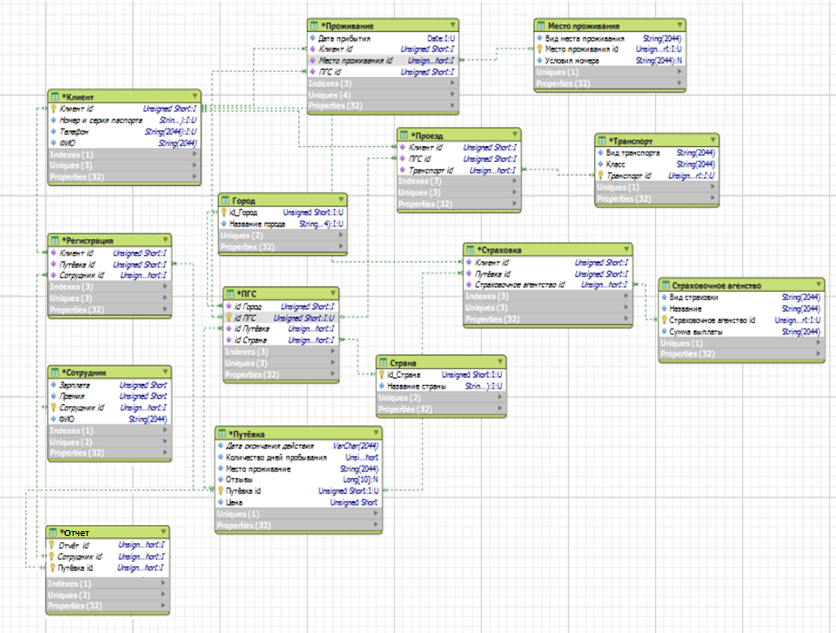


Рис. 2.1 – ER-диаграмма

Условные обозначения:

I – Index,

U–Unique.

# ВЫБОР ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В качестве языка программирования выбран Phyton для программирования сервера и Html, CSS, JS для верстки сайта.

В качестве СУБД была выбрана PostgreSQL. Данная СУБД обладает рядом преимуществ:

1. PostgreSQL объектно-реляционная СУБД. Это даёт ему некоторые преимущества над другими SQL базами данных с открытым исходным кодом;
2. существует возможность расширения функционала за счет использования хранимых процедур, триггеров, представлений, составных типов данных;
3. существует довольно большое сообщество, в котором можно запросто найти ответы на различные вопросы, а также хорошо проработанная документация;
4. PostgreSQL – бесплатная СУБД с открытым кодом;
5. существует множество API под разные платформы.
6. Postgresql представляет из себя объектно-реляционную базу данных, которая работает только на одном движке - storage engine. Все таблицы представлены в виде объектов, они могут наследоваться, а все действия с таблицами выполняются с помощью объективно ориентированных функций.

Во время разработки использовались следующие фреймворки:

1. Django - свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC. Проект поддерживается организацией Django Software Foundation. Сайт на Django строится из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми;
2. Psycopg2 – библиотека классов, необходимая для работы с СУБД PostgreSQL.

# СОЗДАНИЕ БАЗЫ ДАННЫХ

Запрос на создание одной из таблиц БД - Tour. Данный запрос отвечает за создание таблицы и её атрибутов.

CREATE TABLE "Tour"

("Tour\_id" serial NOT NULL PRIMARY KEY,

"Termination\_date" DATE ,

"Amount\_of\_tour\_days" INTEGER NOT NULL,

"Description" TEXT,

"Price" INTEGER NOT NULL CHECK("Price" > 0)

);

В скобках указываются через запятую названия атрибутов и их типы данных. В данном случае столбец Termination\_date представляет из себя дату, Reviews, Description имеют текстовое значение с типом TEXT, а Price, Amount\_of\_tour\_days представляют из себя число с типом INTEGER. В данной таблице некоторым атрибутам присвоено значение NOT NULL, которое означает, что кортеж не может быть создан, пока эти значения не будут заполнены. PRIMARYKEY("Id") означает, что поле Id это первичный ключ данной таблицы. Поле CHECK является полем для проверки, которую указывают в круглых скобках.

Ещё один запрос на создание – Registration:

CREATE TABLE "Registration"(

"Tour\_id" INTEGER NOT NULL,

"Client\_id" INTEGER NOT NULL,

"Employee\_id" INTEGER NOT NULL,

FOREIGN KEY ("Tour\_id") REFERENCES "Tour"

ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY ("Client\_id") REFERENCES "Client"

ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY ("Employee\_id") REFERENCES "Employee"

ON UPDATE CASCADE ON DELETE CASCADE,

PRIMARY KEY("Client\_id", "Tour\_id", "Employee\_id")

);

В данном случае столбец Tour\_id представляет из себя целое число, которое является внешним ключом на таблицу Tour, аналогично для Client\_id и Employee\_id. Запись PRIMARY KEY("Client\_id", "Tour\_id", "Employee\_id") означает, что первичный ключ данной таблицы является составным, состоящий из трёх атрибутов – Client\_id, Tour\_id и Employee\_id. Запись ON UPDATE CASCADE означает, что при обновлении первичного ключа данный внешний ключ также приобретёт новое значение, равное новому значению первичного ключа других таблиц. ON DELETE CASCADE – при удалении записи из таблицы Tour все кортежи, которые ссылались на удалённую запись, также каскадно удалятся.

Все остальные таблицы созданы с использованием соответствующих запросов CREATE TABLE, приведенных в Приложении Б.

# РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БАЗЕ ДАННЫХ

В данной ПрО присутствует 3 роли. Подробнее с точки зрения прав на таблицы БД роли это:

1. Администратор–имеет возможность выполнять все CRUD операции над сущностями Сотрудник, имеет права на чтение, запись и редактирование сущностей Тур, Отель, Транспорт, Город, Страна и связующие таблицы.
2. Сотрудник – может производить чтение сущности Тур. Удаление, чтение и редактирование сущности Клиент.
3. Клиент – может производить чтение сущности Тур, подавать запрос на регистрацию на определённый тур, оставлять отзыв о туре.

Все запросы на создание ролей можно посмотреть в приложении Г.

Ранее было сказано, что администратор добавляет сотрудников, для этого необходимо использовать хранимую процедуру на создание сотрудника, так как именно эта роль представляет собой наибольший интерес с точки зрения сложности запроса.

CREATE OR REPLACE FUNCTION create\_employee(employee\_login VARCHAR, employee\_password VARCHAR)

RETURNS VOID

AS $$

BEGIN

EXECUTE 'CREATE USER "'|| employee\_login ||'" WITH PASSWORD

"'|| employee\_password ||'";';

EXECUTE 'GRANT SELECT

ON ALL TABLES IN SCHEMA public

TO "'|| employee\_login ||'";';

EXECUTE 'GRANT ALL

ON TABLE "Clients", "ClientTour"

TO "'|| employee\_login ||'";';

EXECUTE 'GRANT USAGE

ON ALL SEQUENCES IN SCHEMA public

TO "'|| employee\_login ||'";';

EXECUTE 'GRANT ALL

ON ALL FUNCTIONS IN SCHEMA public

TO "'|| employee\_login ||'";';

END;

$$ LANGUAGE plpgSQL;

Заголовок функции – CREATE OR REPLACE FUNCTION create\_employee(employee\_login VARCHAR, employee\_password VARCHAR) RETURNS VOID означает, что будет создана или заменена (при наличии) функция с названием create\_employee, которая принимает два параметра: логин и пароль. Данная функция не будет ничего возвращать. AS $$ BEGIN – начало тела функции, а END $$ – конец тела функции.

EXECUTE 'CREATE USER " '|| employee\_login ||' "

WITH PASSWORD '' '|| employee\_password ||' '';' - означает создание пользователя с заданным логином и паролем, тут использовался динамический SQL поскольку логин и пароль передаются в качестве параметров и с помощью простого языка SQL этого добиться не выйдет, так же динамический SQL используется с целью учёта регистра логина. Для использования динамического SQL существует команда EXECUTE, после которой в одинарных ковычках текст на языке plpgSQL. Команда CREATE USER создаёт пользователя, WITH PASSWORD указывает, что данный пользователь будет иметь свой пароль. Стоит упомянуть, что команда CREATE USER аналогична команде CREATE ROLE LOGIN, что даёт роли право на авторизацию в БД.

Далее идет команда - EXECUTE 'GRANT ALL ON ALL FUNCTIONS IN SCHEMA public TO " '|| employee\_login ||' ";';

Команда GRANT используется, когда нужно наделить роль какими-то правами. В данном случае это наделение роли правом на выполнение всех операций над всеми функциями в схеме public.

Команда EXECUTE 'GRANT USAGEONALL SEQUENCES IN SCHEMA public TO " '|| employee\_login ||' ";' означает, что роль наделена правами доступа ко всем последовательностям в БД.

Полный список всех остальных хранимых процедур можно посмотреть в Приложении В.

Ниже, в таблице 5.1, приведена информация о правах каждой роли на каждую таблицу, где столбцы – роли, а строки – таблицы в БД, а ячейки – имеющиеся CRUD операции.

|  |  |  |  |
| --- | --- | --- | --- |
| Роль\Таблица | Администратор | Сотрудник | Клиент |
| City | CRUD | R | R |
| Country | CRUD | R | R |
| Transport | CRUD | R | R |
| Place\_of\_residence | CRUD | R | R |
| Insurance\_agency | CRUD | R | - |
| Employee | CRUD | R | R |
| Client | R | CRUD | - |
| Tour | CRUD | R | R |
| Registration | R | CRUD | - |
| TCC | CRUD | R | R |
| Insurance | R | CRUD | - |
| Passage | R | CRUD | - |
| Residence | R | CRUD | - |

Таблица 5.1 – Права ролей на таблицы в БД

# ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Для реализации информационной системы написано несколько py скриптов, с разделением по слоям и ответственности.

Рассмотрим основные скрипты каждого слоя :

1. В роли DAL выступает .py скрипт views. Скрипт используется для обработки GET или POST запросов и отправку ответа на клиент.
2. В качестве BLL выступают классы написанные на Python. Каждый класс, такой как Client, Employee, Tour, City, Registration представляет собой часть информационной системы. Так как ответ от сервера БД оформляется в виде 2-мерного массива, то нам не требуется никакая дополнительная обработка.
3. tamplates – main.html, sign\_in.html, sign\_up.html, search.html, nav.html, admin\_page.html

Рисунок 7.1 - Диаграма классов системы «Туристическое агентство»

Рассмотрим процесс авторизации и других view

from django.http import HttpResponse,JsonResponse

from django.shortcuts import render, redirect

from django.contrib.auth import logout, login, authenticate

from django.contrib.auth.decorators import login\_required, permission\_required

from django.contrib.auth.models import User

from django.contrib.messages import error as ms\_error

from django.views.generic.list import ListView

import datetime

from . import rootmodify as rm

def main\_page(request, \*args, \*\*kwargs):

context={}

return render(request,"main.html",context)

def register\_user\_view(request):#отрисовка страницы регистрации, регистрация

if request.method == "POST": #если пост, значит данные о пользователе введены, создаем и выполняем вход

username = request.POST['username']

mail = request.POST['email']

password = request.POST['password']

if mail != "":

user = User.objects.create\_user(username, email = mail, password = password)

else:

user = User.objects.create\_user(username, password = password)

user.save()

user = authenticate(request, username=username, password=password)

if user is not None:

login(request, user)

else:

ms\_error(request, ('Your credentials are wrong. Sorry.'))

# = request.REQUEST.get(REDIRECT\_FIELD\_NAME, reverse('frontend:index'))

return redirect('main')

else: #иначе отрисовываем форму создания

return render (request, 'sign\_up.html',{})

def sign\_in\_view(request):#отрисовка страницы входа

#request.path = ''

return render(request, 'sign\_in.html')

def loginer\_view(request):#обработка входа

""" Вызывает соответствующее представление Django, анализирует результат. В случае ошибки генерирует сообщение и возвращает пользователя на прежнюю страницу. """

username = request.POST['username']

password = request.POST['password']

user = authenticate(request, username=username, password=password)

if user is not None:

login(request, user)

else:

ms\_error(request, ('Your credentials are wrong. Sorry.'))

# = request.REQUEST.get(REDIRECT\_FIELD\_NAME, reverse('frontend:index'))

if not request.user.is\_authenticated:

context = {

'message': "Пользователь с такими логином и паролем не зарегистрирован"

}

return render(request, 'sign\_in.html', context)

if request.user.groups.filter(name='admin').exists():

return redirect('admin\_page')

return redirect('main')

def logout\_view(request):#обработка выхода

print ("User is trying to logout")

logout(request)

return redirect('main')

def admin\_page\_view(request):#админ панель

if not request.user.groups.filter(name='admin').exists():

return redirect('main')

return render(request, 'admin\_page.html')

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

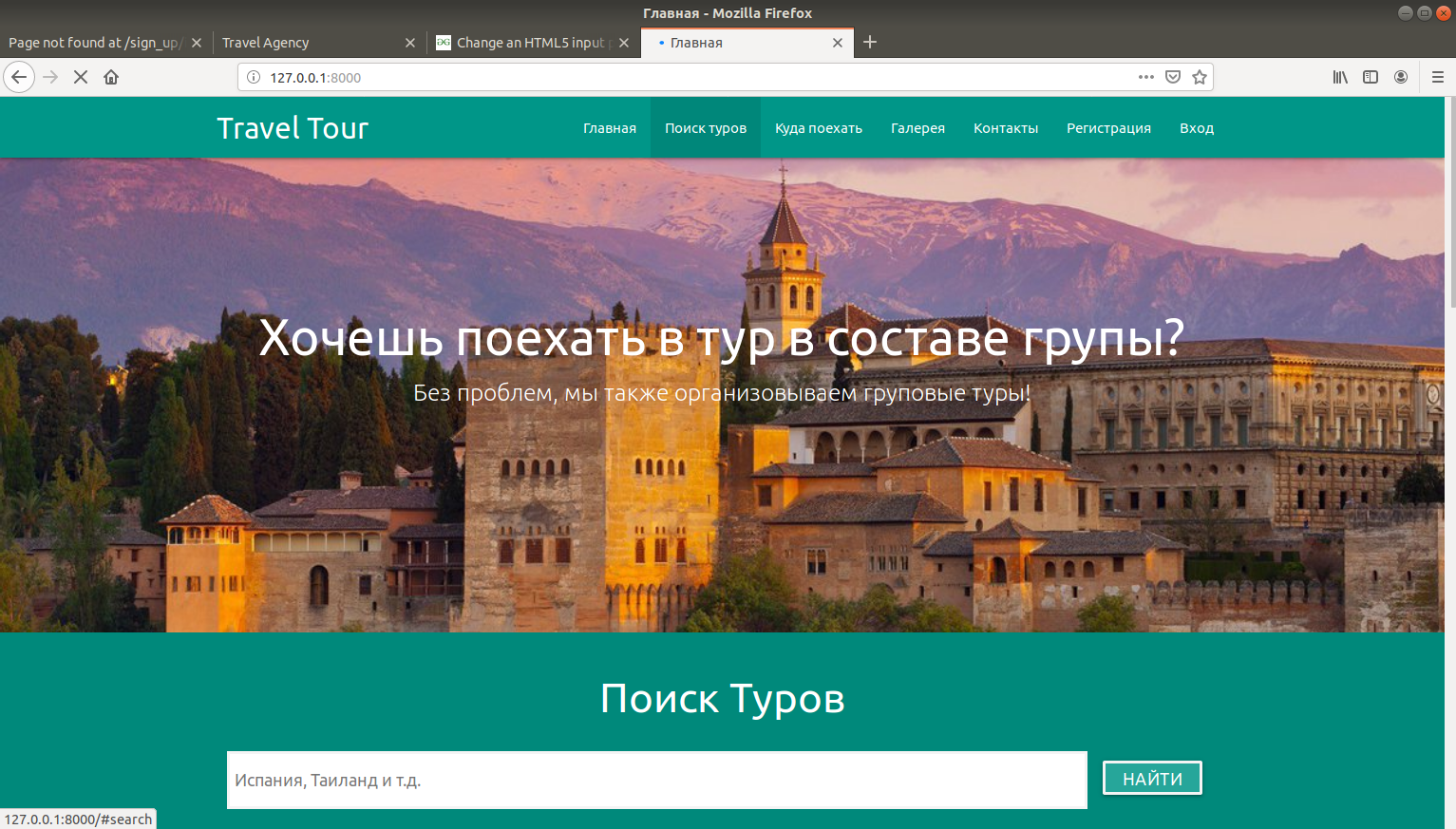
При запуске программы перед пользователем появляется главная страничка сайта (рис. 8.1).

Рисунок 8.1 – Главная страничка сайта

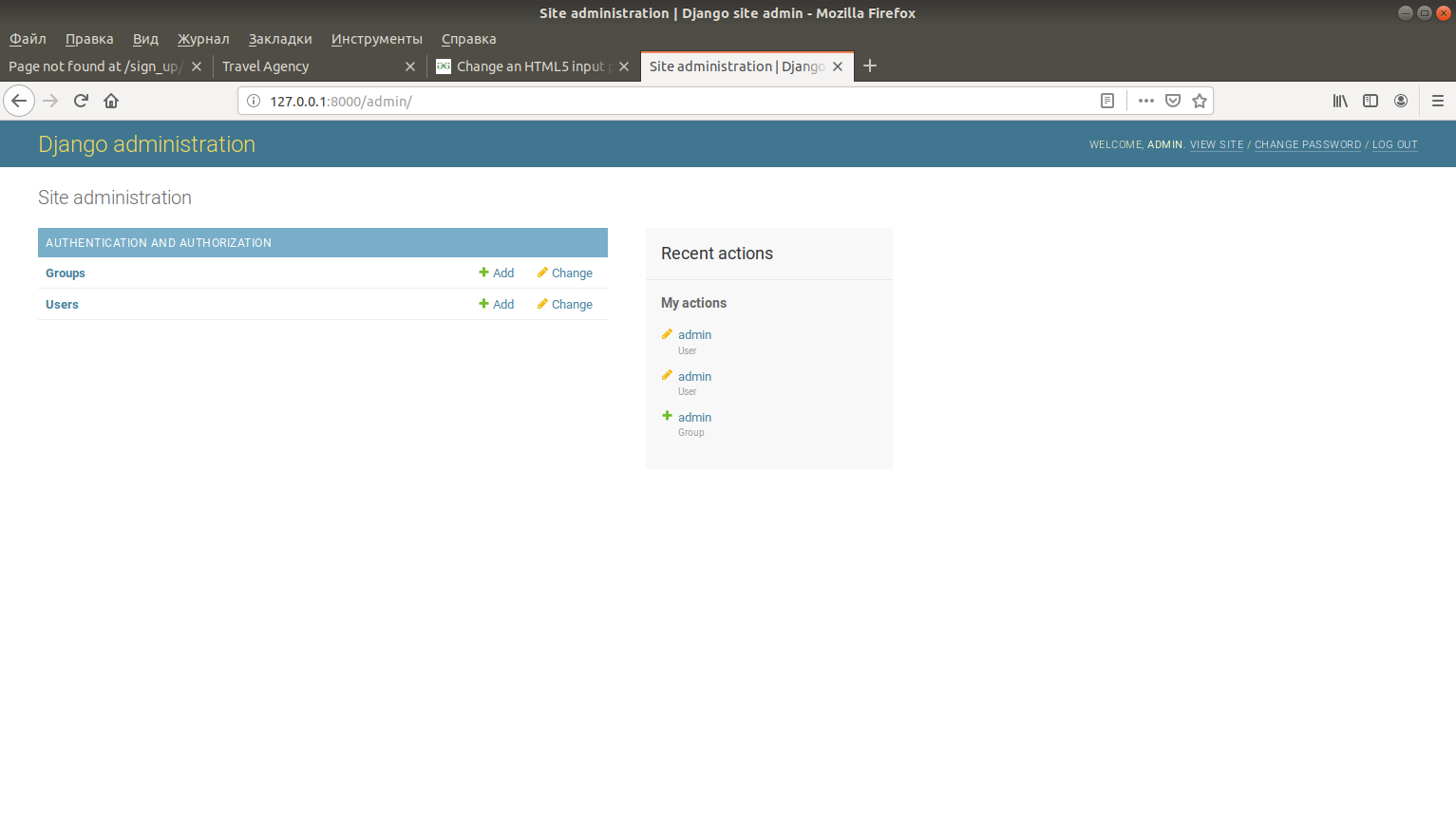
Для некоторых ролей в данной информационной системе существуют складка в главном меню админ панель, которая выполняет задачи этих пользователей данной информационной системе.

Рисунок 8.2 – Админ панель для администратора

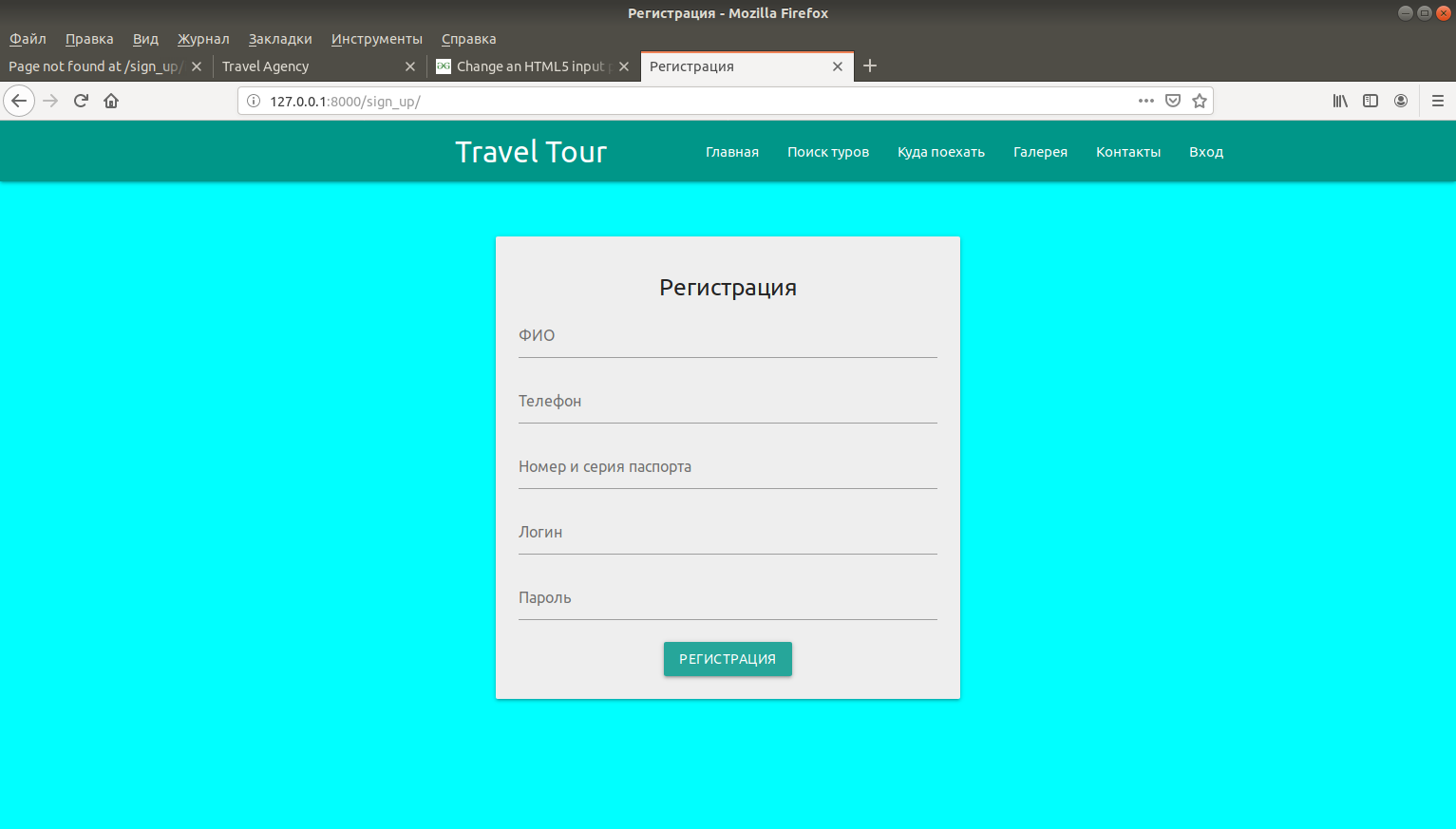
Также есть вкладки Регистрация и Вход для авторизации пользователей

Рисунок 8.3 – Вкладка Регистрация пользователя

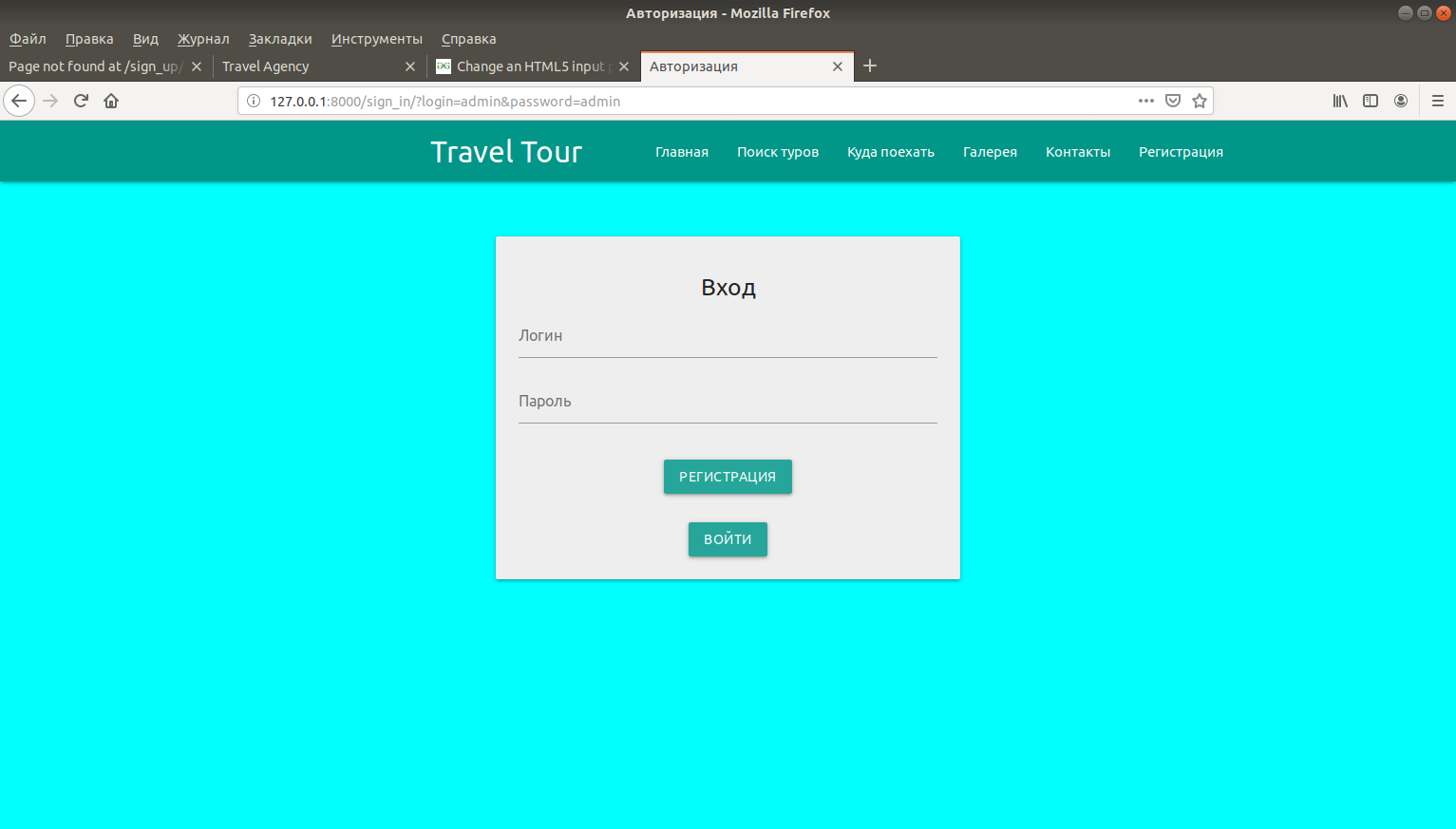


Рисунок 8.3 – Вкладка Вход

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы сформулированы цели создания приложения для туристического агентства, определен перечень пользователей (администратор, сотрудник, менеджер) и задач, которые они решают в рассматриваемой предметной области. Определены требования к данным и спроектирована база данных из 16 таблиц для использования в приложении.

Для создания приложения выбраны трёхуровневая архитектура клиент-сервер, шаблон проектирования MVC, СУБД PostgreSQL, язык программирования Phyton, HTML5, JS, CSS для разработки дружелюбного пользовательского интерфейса.

Создан сайт, который даёт возможность удобно манипулировать данными предметной области. Кроме того, разграничен доступ к приложению со стороны различных категорий пользователей путем использования механизма ролей и привилегий, авторизации.

За счёт использования в качестве архитектуры информационной системы трёхслойной архитектуры была достигнута гибкость и слабая зависимость между слоями приложения, в следствии чего приложение является легко расширяемым. Внедрение новых задач, относящихся к области анализа, не составило бы трудностей. К примеру, можно было бы добавить систему акций, которая помогла бы более выгодно оформлять тур и добавила бы популярности агентству. Приложение также можно улучшить в области пользовательского интерфейса, так как есть очень большое количество фреймворков, а было использовано лишь некоторые.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. UserStories.[Электронный ресурс] – Режим доступа: https://www.mountaingoatsoftware.com/agile/user-stories;
2. Малахов Е.В. Организация баз данных: конспект.– О.: ОНУ, 2015–169 с.
3. Паттерны для новичков: MVCvsMVPvsMVVM. [Электронный ресурс] – Режим доступа: <https://habrahabr.ru/post/215605>
4. Трёхуровневая архитектура [Електронний ресурс] – Режим доступа: <https://ru.wikipedia.org/wiki/Трёхуровневая>\_архитектура
5. Model-View-Controller. [Електронний ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/Model-View-Controller

# ПРИЛОЖЕНИЕ А

СПИСОК ЗАДАЧ ПОЛЬЗОВАТЕЛЕЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ

1. Администратор
2. Добавление сотрудников

Входные данные: ФИО, номер телефона, зарплата, премия.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Редактирование сотрудников

Входные данные: ФИО, номер телефона, зарплата, премия.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Удаление сотрудников

Входные данные: id сотрудника.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось

выполнить;

1. Просмотр информации о созданных турах

Входные данные: отсутствуют

Выходные данные: список всей информации по данному туру;

1. Добавление городов

Входные данные: название города.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Редактирование городов

Входные данные: название города.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Удаление городов

Входные данные: id города.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось

выполнить;

1. Добавление стран

Входные данные: название страны.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Редактирование стран

Входные данные: название страны.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Удаление стран

Входные данные: id страны.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось

выполнить;

1. Добавление мест проживания

Входные данные: название места проживания, тип места проживания, условие номеров.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Редактирование места проживания

Входные данные: название места проживания, тип места проживания, условие номеров.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Удаление места проживания

Входные данные: id места проживания.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось

выполнить;

1. Добавление туров

Входные данные: дата окончания тура, длительность тура, описание тура, цена, максимальное и минимальное количество туристов.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Редактирование туров

Входные данные: дата окончания тура, длительность тура, описание тура, цена, максимальное и минимальное количество туристов.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Удаление тура

Входные данные: id тура.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось

выполнить;

1. Добавление транспорта

Входные данные: тип транспорта, класс транспорта.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Редактирование места транспорта

Входные данные: тип транспорта, класс транспорта.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Удаление места транспорта

Входные данные: id транспорта.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось

выполнить;

1. Добавление страховочного агентства

Входные данные: название страховочного агентства, цена страховки, сумма выплаты.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Редактирование места страховочного агентства

Входные данные: название страховочного агентства, цена страховки, сумма выплаты.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Удаление места страховочного агентства

Входные данные: id страховочного агентства.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось

выполнить;

1. Начислить премию сотруднику

Входные данные: id сотрудника, премия сотрудника.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось

выполнить;

1. Сотрудник
2. Добавление клиентов

Входные данные: ФИО, номер телефона, номер и серия паспорта.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Редактирование клиентов

Входные данные: ФИО, номер телефона, номер и серия паспорта.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Удаление клиентов

Входные данные: id клиента.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Просмотр информации о турах в наличии

Входные данные: отсутствуют

Выходные данные: список всей информации по всем турам;

1. Регистрация клиентов на тур

Входные данные: дата регистрации, id клиента, id тура.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

1. Клиент
   1. Поиск туров по критериям

Входные данные: название страны, название города, минимальная и максимальная цена, длительность поездки.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

* 1. Заказ регистрации на тур

Входные данные: id тура, id места проживания, id транспорта, id страховочного агентства.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

* 1. Заказ обратного звонка

Входные данные: номер телефона.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

* 1. Оставить отзыв о туре

Входные данные: id тура, текст отзыва, оценка.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

* 1. Оставить отзыв о сотруднике

Входные данные: id сотрудника, текст отзыва, оценка.

Выходные данные: Результат проведения операции или ошибка, если запрос не удалось выполнить;

# ПРИЛОЖЕНИЕ Б

ЗАПРОСЫ НА СОЗДАНИЕ БАЗЫ ДАННЫХ

CREATE TABLE "Client"

("Client\_id" serial NOT NULL PRIMARY KEY,

"Passport\_number\_and\_series" VARCHAR(8) NOT NULL,

"Phone\_number" VARCHAR(13) UNIQUE,

"First\_name" varchar NOT NULL,

"Last\_name" varchar NOT NULL,

"Patronymic" varchar NOT NULL

);

CREATE TABLE "Employee"

("Employee\_id" serial NOT NULL PRIMARY KEY,

"Salary" integer NOT NULL CHECK("Salary" > 0),

"Phone\_number" VARCHAR(13) UNIQUE,

"First\_name" varchar NOT NULL,

"Last\_name" varchar NOT NULL,

"Patronymic" varchar NOT NULL,

"Award" integer CHECK("Award" > 0)

);

CREATE TABLE "Tour"

("Tour\_id" serial NOT NULL PRIMARY KEY,

"Termination\_data" DATE ,

"Amount\_of\_tour\_days" integer NOT NULL,

"Description" text,

"Price" integer NOT NULL CHECK("Price" > 0)

"Max\_trevelers" integer NOT NULL CHECK("Max\_travelers" > 0)

);

CREATE TABLE "Registration"

("Registration\_data" DATE ,

"Client\_id" integer NOT NULL,

"Tour\_id" integer NOT NULL,

"Employee\_id" integer NOT NULL,

FOREIGN KEY ("Client\_id") REFERENCES "Client" ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY ("Tour\_id") REFERENCES "Tour" ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY ("Employee\_id") REFERENCES "Employee" ON UPDATE CASCADE ON DELETE CASCADE,

PRIMARY KEY("Client\_id", "Tour\_id", "Employee\_id")

);

CREATE TABLE "City"

("City\_id" serial NOT NULL PRIMARY KEY,

"City\_name" varchar NOT NULL

);

CREATE TABLE "Country"

("Country\_id" serial NOT NULL PRIMARY KEY,

"Country\_name" varchar NOT NULL

);

CREATE TABLE "TCC"

("TCC\_id" serial NOT NULL,

"City\_id" integer NOT NULL,

"Country\_id" integer NOT NULL,

"Tour\_id" integer NOT NULL,

FOREIGN KEY ("City\_id") REFERENCES "City" ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY ("Tour\_id") REFERENCES "Tour" ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY ("Country\_id") REFERENCES "Country" ON UPDATE CASCADE ON DELETE CASCADE,

PRIMARY KEY("TCC\_id"),

UNIQUE("City\_id", "Tour\_id", "Country\_id")

);

CREATE TABLE "Place\_of\_residence"

("Place\_of\_residence\_id" serial NOT NULL PRIMARY KEY,

"Name\_of\_place\_of\_residence " varchar NOT NULL,

"Type\_of\_place\_of\_residence" varchar NOT NULL CHECK("Type\_of\_place\_of\_residence" IN('Hotel','Motel','Hostel', 'GrandHotel')),

"Conditions" varchar NOT NULL

);

CREATE TABLE "Transport"

("Transport\_id" serial NOT NULL PRIMARY KEY,

"Class\_of\_transport" varchar NOT NULL CHECK ("Class\_of\_transport" IN('Business','CV','Coupe','Econom')),

"Type\_of\_transport" varchar NOT NULL

);

CREATE TABLE "Insurance\_agency"

("Insurance\_agency\_id" serial NOT NULL PRIMARY KEY,

"Name\_of\_insurance\_agency" varchar NOT NULL,

"Price\_of\_insurance" integer NOT NULL CHECK("Price\_of\_insurance" > 0),

"Payout\_amount" integer NOT NULL CHECK("Payout\_amount" > 0)

);

CREATE TABLE "Insurance"

("Client\_id" integer NOT NULL,

"Tour\_id" integer NOT NULL,

"Insurance\_agency\_id" integer NOT NULL,

FOREIGN KEY ("Client\_id") REFERENCES "Client" ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY ("Tour\_id") REFERENCES "Tour" ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY ("Insurance\_agency\_id") REFERENCES "Insurance\_agency" ON UPDATE CASCADE ON DELETE CASCADE,

PRIMARY KEY("Client\_id", "Tour\_id", "Insurance\_agency\_id")

);

CREATE TABLE "Passage"

("Client\_id" integer NOT NULL,

"TCC\_id" integer NOT NULL,

"Transport\_id" integer NOT NULL,

FOREIGN KEY ("Client\_id") REFERENCES "Client" ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY ("TCC\_id") REFERENCES "TCC" ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY ("Transport\_id") REFERENCES "Transport" ON UPDATE CASCADE ON DELETE CASCADE,

PRIMARY KEY("Client\_id", "TCC\_id", "Transport\_id")

);

CREATE TABLE "Residence"

("Ariving\_data" DATE ,

"Client\_id" integer NOT NULL,

"TCC\_id" integer NOT NULL,

"Place\_of\_residence\_id" integer NOT NULL,

FOREIGN KEY ("Client\_id") REFERENCES "Client" ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY ("TCC\_id") REFERENCES "TCC" ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY ("Place\_of\_residence\_id") REFERENCES "Place\_of\_residence" ON UPDATE CASCADE ON DELETE CASCADE,

PRIMARY KEY("Client\_id", "TCC\_id", "Place\_of\_residence\_id")

);

CREATE TABLE "Report"

("Report\_id" serial NOT NULL PRIMARY KEY,

"Employee\_id" integer NOT NULL,

"Tour\_id" integer NOT NULL,

"Amount\_of\_sellings" intege,

FOREIGN KEY ("Employee\_id") REFERENCES "Employee" ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY ("Tour\_id") REFERENCES "Tour" ON UPDATE CASCADE ON DELETE CASCADE,

UNIQUE("Employee\_id", "Tour\_id")

);

CREATE TABLE "Tour\_Review"

("Review\_id" serial NOT NULL PRIMARY KEY,

"Client\_id" integer NOT NULL,

"Tour\_id" integer NOT NULL,

"Text" varchar NOT NULL,

"Mark" integer NOT NULL CHECK("Mark" > 0 AND "Mark" < 6),

FOREIGN KEY ("Client\_id") REFERENCES "Client" ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY ("Tour\_id") REFERENCES "Tour" ON UPDATE CASCADE ON DELETE CASCADE,

UNIQUE("Client\_id", "Tour\_id")

);

CREATE TABLE "Employee\_Review"

("Review\_id" serial NOT NULL PRIMARY KEY,

"Client\_id" integer NOT NULL,

"Employee\_id" integer NOT NULL,

"Text" varchar NOT NULL,

"Mark" integer NOT NULL CHECK("Mark" > 0 AND "Mark" < 6),

FOREIGN KEY ("Client\_id") REFERENCES "Client" ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY ("Employee\_id") REFERENCES "Employee" ON UPDATE CASCADE ON DELETE CASCADE,

UNIQUE("Employee\_id", "Client\_id")

);

CREATE TABLE "Registration\_request"

("Registration\_request\_id" serial NOT NULL PRIMARY KEY,

"Client\_id" integer NOT NULL,

"Tour\_id" integer NOT NULL,

FOREIGN KEY ("Client\_id") REFERENCES "Client" ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY ("Tour\_id") REFERENCES "Tour" ON UPDATE CASCADE ON DELETE CASCADE,

UNIQUE("Tour\_id", "Client\_id")

);

CREATE TABLE "Call\_request"

("Call\_request\_id" serial NOT NULL PRIMARY KEY,

"Client\_id" integer NOT NULL UNIQUE,

"Phone\_number" varchar(13) NOT NULL UNIQUE,

FOREIGN KEY ("Client\_id") REFERENCES "Client" ON UPDATE CASCADE ON DELETE CASCADE

);

# ПРИЛОЖЕНИЕ В

ЗАПРОСЫ К БД ДЛЯ РЕШЕНИЯ ПОСТАВЛЕННЫХ ЗАДАЧ

1. Добавление сотрудников

INSERT INTO "Employee" VALUES(Salary, Phone\_number, First\_name, Last\_name, Patronymic, Award);

1. Редактирование сотрудника

CREATE OR REPLACE FUNCTION employee\_update( employee\_id INT, Salary INT, Phone\_number VARCHAR(13), First\_name varchar, Last\_name varchar, Patronymic varchar, Award INT)

AS $$

DECLARE

f\_employee\_id "Employee"."Employee\_id"%TYPE;

BEGIN

SELECT "Employee\_id" INTO f\_employee\_id FROM "Employee" WHERE "Employee\_id"= employee\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'ERROR: this employee % is not exist', employee\_id;

END IF;

UPDATE TABLE "Employee"

SET "Salary" = Salary, Phone\_number = Phone\_number, First\_name = First\_name, Last\_name = Last\_name, Patronymic = Patronymic, Award = Award

WHERE "Emloyee\_id" = f\_emloyee\_id;

END;

$$ LANGUAGE plpgSQL

1. Удаление сотрудника

DELETE FROM "Employee" WHERE "Employee\_id" = Employee\_id;

1. Просмотр информации о туре

CREATE OR REPLACE FUNCTION tour\_information( Tour\_id INT) RETURNS TABLE("Tour\_id" INT, "City" VARCHAR, "Insuranse\_agency" CHAR, "Payout\_amount" INT, "Amount\_of\_tour\_days" INT, "Price" INT, "Transport\_type" VARCHAR, "Place\_of\_residence\_name" VARCHAR)

AS $$

BEGIN

RETURN QUERY

SELECT T."Tour\_id", C."City\_name", I."Name\_of\_insurance\_agency", I."Payout\_amount", T."Amount\_of\_tour\_days", T."Price", Tr."Type\_of\_transport", P."Name\_of\_place\_of\_residence"

FROM "Tour" AS T JOIN "TCC" USING("Tour\_id")

JOIN "City" AS C USING("City\_id")

JOIN "Insurance" USING("Tour\_id")

JOIN "Insurance\_agency" AS I USING("Insurance\_agency\_id")

JOIN "Passage" USING("TCC\_id")

JOIN "Transport" AS Tr USING("Transport\_id")

JOIN "Residence" USING("TCC\_id")

JOIN "Place\_of\_residence" AS P USING("Place\_of\_residence\_id")

WHERE T."Tour\_id" = Tour\_id

GROUP BY T."Tour\_id", C."City\_name", I."Name\_of\_insurance\_agency", I."Payout\_amount", T."Amount\_of\_tour\_days", T."Price", Tr."Type\_of\_transport", P."Name\_of\_place\_of\_residence";

END;

$$ LANGUAGE plpgSQL;

1. Добавление городов

INSERT INTO "City" VALUES(City\_name);

1. Редактирование городов

CREATE OR REPLACE FUNCTION city\_update( city\_id int, City\_name varchar)AS $$

DECLARE

f\_city\_id "City"."City\_id"%TYPE;

BEGIN

SELECT "City\_id" INTO f\_city\_id FROM "City" WHERE "City\_id"= city\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'ERROR: this city % is not exist', city\_id;

END IF;

UPDATE TABLE "City"

SET "City\_name" = City\_name

WHERE "City\_id" = f\_city\_id;

END;

$$ LANGUAGE plpgSQL

1. Удаление города

DELETE FROM "City" WHERE "City\_id" = City\_id;

1. Добавление стран

INSERT INTO "Country" VALUES(country\_name);

1. Редактирование стран

CREATE OR REPLACE FUNCTION country\_create( country\_id int, country\_name varchar)AS $$

DECLARE

f\_city\_id "Country"."Country\_id"%TYPE;

BEGIN

SELECT "Country\_id" INTO f\_country\_id FROM "Country" WHERE "Country\_id"= country\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'ERROR: this country % is not exist', country\_id;

END IF;

UPDATE TABLE "Country"

SET "Country\_name" = country\_name

WHERE "Country\_id" = f\_country\_id;

END;

$$ LANGUAGE plpgSQL

1. Удаление стран

DELETE FROM "Country" WHERE "Country\_id" = Country\_id;

1. Редактирование места проживания

CREATE OR REPLACE FUNCTION place\_of\_residence\_create( place\_of\_residence\_id int, place\_of\_residence\_name varchar, type\_of\_place\_of\_residence varchar, conditions varchar)

AS $$

DECLARE

f\_place\_of\_residence\_id "Place\_of\_residence"."Place\_of\_residence\_id"%TYPE;

BEGIN

SELECT "Place\_of\_residence\_id" INTO f\_place\_of\_residence\_id FROM "Place\_of\_residence" WHERE "Place\_of\_residence\_id"= place\_of\_residence\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'ERROR: this place\_of\_residence % is not exist', place\_of\_residence\_id;

END IF;

UPDATE TABLE "Place\_of\_residence"

SET "Place\_of\_residence\_name" = place\_of\_residence\_name, "Type\_of\_place\_of\_residence" = type\_of\_place\_of\_residence, "Conditions" = conditions

WHERE "Place\_of\_residence\_id" = f\_place\_of\_residence\_id;

END;

$$ LANGUAGE plpgSQL

1. Редактирование тура

CREATE OR REPLACE FUNCTION tour\_create( place\_of\_residence\_id int, Termination\_data DATE, Amount\_of\_tour\_days INT, Description Text, Price INT)

AS $$

DECLARE

f\_tour\_id "Tour"."Tour\_id"%TYPE;

BEGIN

SELECT "Tour\_id" INTO f\_tour\_of\_residence\_id FROM "Tour\_of\_residence" WHERE "Tour\_id"= tour\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'ERROR: this tour % is not exist', tour\_id;

END IF;

UPDATE TABLE "Tour"

SET "Termination\_data" = Termination\_data, "Amount\_of\_tour\_days" = Amount\_of\_tour\_days, "Description" = description, "Price" = Price

WHERE "Tour\_id" = f\_tour\_id;

END;

$$ LANGUAGE plpgSQL

1. Удаление тура

CREATE OR REPLACE FUNCTION tour\_delete( Tour\_id INT)

AS $$

BEGIN

DELETE FROM "Tour" WHERE "Tour\_id" = Tour\_id;

END;

$$ LANGUAGE plpgSQL;

1. Редактирование транспорта

CREATE OR REPLACE FUNCTION transport\_create( transport\_id int, class\_of\_transport varchar, type\_of\_transport varchar)

AS $$

DECLARE

f\_transport\_id "Transport"."Transport\_id"%TYPE;

BEGIN

SELECT "Transport\_id" INTO f\_transport\_id FROM "Transport" WHERE "Transport\_id"= transport\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'ERROR: this transport % is not exist', transport\_id;

END IF;

UPDATE TABLE "Transport"

SET "Type\_of\_transport" = type\_of\_transport, "Class\_of\_transport" = class\_of\_transport;

WHERE "Transport\_id" = f\_transport\_id;

END;

$$ LANGUAGE plpgSQL

1. Редактирование страховочного агентства

CREATE OR REPLACE FUNCTION insurance\_agency\_create( insurance\_agency\_id int, Name\_of\_insurance\_agency varchar, Price\_of\_insurance INT, Payout\_amount INT)

AS $$

DECLARE

f\_insurance\_agency\_id "Insurance\_agency"."Insurance\_agency\_id"%TYPE;

BEGIN

SELECT "Insurance\_agency\_id" INTO f\_insurance\_agency\_id FROM "Insurance\_agency" WHERE "Insurance\_agency\_id"= insurance\_agency\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'ERROR: this insurance\_agency % is not exist', insurance\_agency\_id;

END IF;

UPDATE TABLE "Insurance\_agency"

SET "Name\_of\_insurance\_agency" = Name\_of\_insurance\_agency, "Price\_of\_insurance" = Price\_of\_insurance, "Payout\_amount" = Payout\_amount;

WHERE "Insurance\_agency\_id" = f\_insurance\_agency\_id\_id;

END;

$$ LANGUAGE plpgSQL

1. Начислить премию сотруднику

CREATE OR REPLACE FUNCTION give\_award\_to\_employee( employee\_id int, Award INT)

AS $$

DECLARE

f\_employee\_id "Employee"."Employee\_id"%TYPE;

BEGIN

SELECT "Employee\_id" INTO f\_employee\_id FROM "Employee" WHERE "Employee\_id"= employee\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'ERROR: this employee % is not exist', employee\_id; END IF;

UPDATE TABLE "Employee"

SET "Award" = Award;

WHERE "Employee\_id" = f\_employee\_id;

END;

$$ LANGUAGE plpgSQL

1. Редактирование клиента

CREATE OR REPLACE FUNCTION client\_update( client\_id INT, Passport\_number\_and\_series VARCHAR(8), Phone\_number VARCHAR(13), First\_name varchar, Last\_name varchar NOT NULL, Patronymic varchar NOT NULL)

AS $$

DECLARE

f\_client\_id "Client"."Client\_id"%TYPE;

BEGIN

SELECT "Client\_id" INTO f\_client\_id FROM "Client" WHERE "Client\_id"= client\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'ERROR: this client % is not exist', client\_id;

END IF;

UPDATE TABLE "Client"

SET "Passport\_number\_and\_series" = Passport\_number\_and\_series, Phone\_number = Phone\_number, First\_name = First\_name, Last\_name = Last\_name, Patronymic = Patronymic;

WHERE "Client\_id" = f\_client\_id;

END;

$$ LANGUAGE plpgSQL

1. Просмотр информации о туре

CREATE OR REPLACE FUNCTION tour\_information() RETURNS TABLE("Tour\_id" INT, "Country" VARCHAR, "City" VARCHAR, "Insuranse\_agency" CHAR, "Payout\_amount" INT, "Amount\_of\_tour\_days" INT, "Price" INT, "Transport\_type" VARCHAR, "Place\_of\_residence\_name" VARCHAR)

AS $$

BEGIN

RETURN QUERY

SELECT T."Tour\_id", Cou."Country\_name", C."City\_name", I."Name\_of\_insurance\_agency", I."Payout\_amount", T."Amount\_of\_tour\_days", T."Price", Tr."Type\_of\_transport", P."Name\_of\_place\_of\_residence"

FROM "Tour" AS T JOIN "TCC" USING("Tour\_id")

JOIN "City" AS C USING("City\_id")

JOIN "Country" AS Cou USING("Country\_id")

JOIN "Insurance" USING("Tour\_id")

JOIN "Insurance\_agency" AS I USING("Insurance\_agency\_id")

JOIN "Passage" USING("TCC\_id")

JOIN "Transport" AS Tr USING("Transport\_id")

JOIN "Residence" USING("TCC\_id")

JOIN "Place\_of\_residence" AS P USING("Place\_of\_residence\_id")

GROUP BY T."Tour\_id", Cou."Country\_name", C."City\_name", I."Name\_of\_insurance\_agency", I."Payout\_amount", T."Amount\_of\_tour\_days", T."Price", Tr."Type\_of\_transport", P."Name\_of\_place\_of\_residence";

END;

$$ LANGUAGE plpgSQL;

1. Регистрация клиента на тур

CREATE OR REPLACE FUNCTION registration\_create( registration\_date DATE, client\_id INT, tour\_id INT, employee\_id INT) RETURNS INTEGER

AS $$

DECLARE

f\_client\_id "Client"."Client\_id"%TYPE;

f\_employee\_id "Employee"."Employee\_id"%TYPE;

f\_tour\_id "Tour"."Tour\_id"%TYPE;

BEGIN

SELECT "Client\_id" INTO f\_client\_id FROM "Client" WHERE "Client\_id"= client\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'ERROR: this client % is not exist', client\_id;

END IF;

SELECT "Employee\_id" INTO f\_employee\_id FROM "Employee" WHERE "Employee\_id"= employee\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'ERROR: this employee % is not exist', employee\_id;

END IF;

SELECT "Tour\_id" INTO f\_tour\_id FROM "Tour" WHERE "Tour\_id"= tour\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'ERROR: this tour % is not exist', tour\_id;

END IF;

INSERT INTO "Registration" VALUES(registration\_date, f\_client\_id, f\_employee\_id, f\_tour\_id);

END;

$$ LANGUAGE plpgSQL;

1. Поск туров по критериям

CREATE OR REPLACE FUNCTION search\_tour( country varchar, price\_low INT, price\_high INT) RETURNS TABLE("Country" VARCHAR, "City" VARCHAR, "Insuranse\_agency" CHAR, "Price\_of\_insurance" INT, "Amount\_of\_tour\_days" INT, "Price" INT, "Transport\_type" CHAR)

AS $$

BEGIN

RETURN QUERY

SELECT Cou."Country\_name", C."City\_name", I."Name\_of\_insurance\_agency", I."Price\_of\_insurance", T."Amount\_of\_tour\_days", T."Price", Tr."Type\_of\_transport"

FROM "Tour" AS T JOIN "TCC" USING("Tour\_id")

JOIN "City" AS C USING("City\_id")

JOIN "Country" AS Cou USING("Country\_id")

JOIN "Insurance" USING("Tour\_id")

JOIN "Insurance\_agency" AS I USING("Insurance\_agency\_id")

JOIN "Passage" USING("TCC\_id")

JOIN "Transport" AS Tr USING("Transport\_id")

WHERE Cou."Country\_name"=country AND T."Price" BETWEEN price\_low AND price\_high AND;

GROUP BY Cou."Country\_name", C."City\_name", I."Name\_of\_insurance\_agency", I."Price\_of\_insurance", T."Amount\_of\_tour\_days", T."Price", Tr."Type\_of\_transport";

END;

$$ LANGUAGE plpgSQL;

1. Заказ регистрации на тур

CREATE OR REPLACE FUNCTION registration\_request(tour\_id INT)

AS $$

DECLARE

f\_tour\_id "Tour"."Tour\_id"%TYPE;

BEGIN

SELECT "Tour\_id" INTO f\_tour\_id FROM "Tour" WHERE "Tour\_id"= tour\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'ERROR: this tour % is not exist', tour\_id;

END IF;

INSERT INTO "Tour\_Request" VALUES(f\_tour\_id);

END;

$$ LANGUAGE plpgSQL;

1. Заказ обратного звонка

CREATE OR REPLACE FUNCTION call\_request(Phone\_number VARCHAR(13))

AS $$

BEGIN

INSERT INTO "Call\_Request" VALUES(Phone\_number);

END;

$$ LANGUAGE plpgSQL;

1. Оставить отзыв о туре

CREATE OR REPLACE FUNCTION tour\_review (Tour\_id INT, text TEXT, mark INT)

AS $$

DECLARE

f\_tour\_id "Tour"."Tour\_id"%TYPE;

BEGIN

SELECT "Tour\_id" INTO f\_tour\_id FROM "Tour" WHERE "Tour\_id"= tour\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'ERROR: this tour % is not exist', tour\_id;

END IF;

INSERT INTO "Tour\_review" VALUES(f\_tour\_id, text, mark);

END;

$$ LANGUAGE plpgSQL;

1. Оставить отзыв о клиенте

CREATE OR REPLACE FUNCTION client\_review (client\_id INT, text TEXT, mark INT)

AS $$

DECLARE

f\_client\_id "Client"."Client\_id"%TYPE;

BEGIN

SELECT "Client\_id" INTO f\_client\_id FROM "Client" WHERE "Client\_id"= client\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'ERROR: this client % is not exist', client\_id;

END IF;

INSERT INTO "Client\_review" VALUES(f\_client\_id, text, mark);

END;

$$ LANGUAGE plpgSQL;

# ПРИЛОЖЕНИЕ Г

ЗАПРОСЫ НА СОЗДАНИЕ ПОЛЬЗОВАТЕЛЕЙ

Создание администратора:

CREATE USER "Admin" WITH PASSWORD 'Admin' CREATE ROLE;

GRANT SELECT

ON ALL TABLES IN SCHEMA public

TO "Admin"

WITH GRANT OPTION;

GRANT ALL

ON TABLE "Client", "City", "Country", "Place\_of\_residance", "Transport", "Tour", "Employee"

TO "Admin"

WITH GRANT OPTION;

GRANT USAGE

ON ALL SEQUENCES IN SCHEMA public

TO "Admin"

WITH GRANT OPTION;

GRANT ALL

ON ALL FUNCTIONS IN SCHEMA public

TO "Admin"

WITH GRANT OPTION;

Создание сотрудника:

CREATE USER "Employee" WITH PASSWORD 'Employee' CREATE ROLE;

GRANT SELECT

ON ALL TABLES IN SCHEMA public

TO "Employee"

WITH GRANT OPTION;

GRANT ALL

ON TABLE "Client", "Registration", "Passage", "Residence"

TO "Employee"

WITH GRANT OPTION;

GRANT USAGE

ON ALL SEQUENCES IN SCHEMA public

TO "Employee"

WITH GRANT OPTION;

GRANT ALL

ON ALL FUNCTIONS IN SCHEMA public

TO "Employee"

WITH GRANT OPTION;

Создание клиента:

CREATE USER "Client" WITH PASSWORD 'Client';

GRANT SELECT

ON TABLE "Tour", "City", "Country", "TCC", "Employee"

TO "Client"

GRANT USAGE ON ALL SEQUENCES IN SCHEMA public

TO "Client";