**tМИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Донецкий национальный технический университет»**

Факультет Интеллектуальных систем и программирования

Кафедра "Программная инженерия" им. Л.П. Фельдмана

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по курсу “Базы данных”

Тема работы: “Хозяйственная деятельность

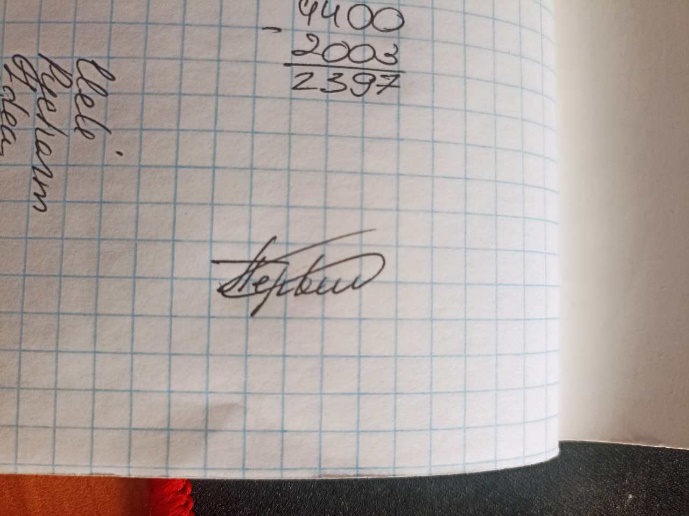
ботанических садов страны”

Руководители:

(подпись) (дата)

Щедрин С.В. Незамова Л.В.

Рычка О. В.

Разработал:

ст. гр. ПИ-21г 13.11.2023Первий Д. Д.

(подпись) (дата)

Донецк – 2023

Министерство образования и науки

Донецкой народной республики

ГОУ ВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Интеллектуальных систем и программирования

Кафедра "Программная инженерия" им. Л.П. Фельдмана

УТВЕРЖДАЮ:

Зав. кафедрой ПИ им. Л.П. Фельдмана

\_\_\_\_\_\_\_\_\_\_\_ проф. Зори С.А.

9.02.2023 г.

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

**на курсовой проект по дисциплине**

**«Базы данных»**

**Тема: «Создание клиент-серверной информационной системы средствами СУБД»**

выдано студенту группы ПИ-21г Первию Денису Дмитриевичу

Руководители курсового проекта: доц. Рычка О.В., ст. пр. Щедрин С.В., асс. Незамова Л.В.

**Содержание задания на курсовой проект**

Описание предметной области:

Разработать БД учета хозяйственной деятельности ботанических садов страны (некоторые статьи расходов). Статьи расходов рассчитать по оплате труда сотрудников (ежемесячно) и содержанию растений (ежемесячно). Один сотрудник может обслуживать несколько растений. Каждое растение содержит паспортные данные о растении и условия ухода за ним. Средства ухода за растениями могут быть штучными (упаковка) или весовыми (кг). Соответственно цена записывается - за упаковку или за 1 кг.

 Необходимо:

1. Спроектировать концептуальную модель базы данных (БД) для заданной предметной области и представить ее в виде взаимосвязанных таблиц, находящихся в третьей нормальной форме (в случае денормализации БД – обосновать необходимость). Выделить базовые таблицы и таблицы-справочники, указать для них первичные и внешние ключи.
2. Создать базу данных в среде СУБД средствами языка SQL. Добавить таблицы, домены, индексы.
3. Разработать не менее шести триггеров (по одному для каждого типа события), как минимум для двух различных таблиц БД. Триггеры типа BEFORE INSERT должны быть созданы для всех таблиц и с использованием генераторов задавать значение первичного ключа для вновь добавляемой записи.
4. Заполнить таблицы БД с использованием соответствующих запросов на языке SQL (не менее десяти записей в каждом справочнике, не менее 10 000 - 50 000 псевдослучайных записей в таблицах).
5. Сформулировать и реализовать следующие виды запросов:

* симметричное внутреннее соединение с условием (два запроса с условием отбора по внешнему ключу, два – по датам);
* симметричное внутреннее соединение без условия (три запроса);
* левое внешнее соединение;
* правое внешнее соединение;
* запрос на запросе по принципу левого соединения;
* итоговый запрос без условия;
* итоговый запрос без условия c итоговыми данными вида: «всего», «в том числе»;
* итоговые запросы с условием на данные (по значению, по маске, с использованием индекса, без использования индекса);
* итоговый запрос с условием на группы;
* итоговый запрос с условием на данные и на группы;
* запрос на запросе по принципу итогового запроса;
* запрос с использованием объединения:
* запросы с подзапросами (с использованием in, not in, case, операциями над итоговыми данными);
* определить процент растений, посаженных до 2010 года по каждому ботаническому саду и в целом по всем садам.
* определить средний возраст и стаж работников по каждому ботаническому саду и по всем садам в целом.
* определить суммарные расходы по уходу за растениями и количество растений по каждому саду.

1. Запросы без параметров реализовать в виде представлений, остальные запросы – в виде хранимых процедур и/или функций. Создать, по меньшей мере, одно модифицируемое представление, используя механизм триггеров. ВСЯ логика проектируемого ПО – на сервере.
2. Разработать клиентское приложение, которое предоставляет следующие возможности для работы с созданной базой данных:

* добавление, редактирование и удаление записей таблиц и модифицируемых представлений;
* работа с наборами данных, находящимися в отношении «один-ко-многим» (создать составную форму для просмотра и редактирования данных родительской и дочерней таблиц);
* поиск и фильтрация данных отображаемых таблиц;
* просмотр результатов выполнения запросов;
* визуализация результатов одного из итоговых запросов (диаграммы, экспорт в Excel).

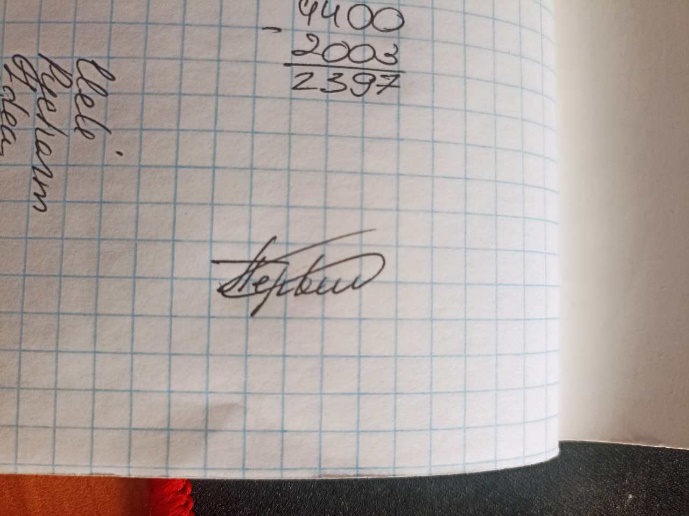
1. Обеспечить защиту данных, информации от несанкционированного доступа

График выполнения курсового проекта

|  |  |
| --- | --- |
| Неделя | Этап выполнения работы |
| 1-2 | Выдача и изучение задания |
| 3 | Анализ требований к системе и способов их реализации |
| 4-5 | Проектирование и реализация БД (таблицы, домены, индексы) |
| 6-7 | Создание триггеров и заполнение таблиц БД |
| 8-9 | Создание представлений и хранимых процедур |
| 10-13 | Разработка клиентского приложения |
| 14 | Тестирование и отладка системы |
| 15 | Оформление пояснительной записки |
| 16-17 | Защита курсового проекта |

Дата выдачи задания: 09.02.2023

Дата защиты курсового проекта:16.05.2023



**Задание принял:** Первий Денис Дмитриевич

**Согласовано:**

Руководители проекта: Щедрин Сергей Валерьевич

Рычка Ольга Валентиновна  
Незамова Лариса Викторовна

РЕФЕРАТ

Пояснительная записка содержит 96 страниц, 84 рисунков, 1 таблицу, 6 источников, 3 приложения.

Объектом исследования данной работы является организация базы данных для поддержки работы ботанических садов страны. Предмет исследования – ботанические сады страны.

Цель работы – разработать приложение учета деятельности ботанических садов страны, приобрести навыки работы с БД и СУБД, научиться проектировать базы данных и составлять запросы на языке структурированных запросов SQL.

Результатом работы является клиентское приложение для учета деятельности ботанических садов страны и реляционная база данных, предназначенная для хранения необходимой для учета деятельности ботанических садов информации. Клиентское приложение и реляционная база данных разработаны в соответствии с требованиями курсового проекта. Разработанная реляционная база данных «Ботанические сады» реализована в СУБД PostgreSQL, клиентское приложение написано на языке программирования C#, серверное приложение реализовано на платформе FastAPI.

БАЗЫ ДАННЫХ, POSTGRESQL, PGADMIN, СУБД, РЕЛЯЦИОННАЯ БАЗА ДАННЫХ, SQL, C#, FASTAPI, PYTHON

СОДЕРЖАНИЕ

[РЕФЕРАТ 6](#_Toc150548253)

[СОДЕРЖАНИЕ 7](#_Toc150548254)

[ВВЕДЕНИЕ 10](#_Toc150548255)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ, ПОСТАНОВКА ЗАДАЧИ 11](#_Toc150548256)

[1.1 Описание входных документов 11](#_Toc150548257)

[1.2 Описание выходных документов 12](#_Toc150548258)

[1.3 Список ограничений 12](#_Toc150548259)

[2 ОБОСНОВАНИЕ ВЫБОРА СУБД POSTGRESQL, ОПИСАНИЕ ВОЗМОЖНОСТЕЙ СУБД POSTGRESQL 13](#_Toc150548260)

[3 ОБОСНОВАНИЕ ВЫБОРА ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ ДЛЯ НАПИСАНИЯ КЛИЕНТСКОЙ ЧАСТИ, ПРОЕКТИРОВАНИЕ СТРУКТУРЫ ПО 14](#_Toc150548261)

[3.1 Не визуальные компоненты 14](#_Toc150548262)

[3.2 Визуальные компоненты отображения данных 15](#_Toc150548263)

[4 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ В ВЫБРАННОЙ СУБД 16](#_Toc150548264)

[4.1 Проектирование концептуальной модели БД 16](#_Toc150548265)

[4.1.1 Инфологическое проектирование 16](#_Toc150548266)

[4.1.2 Даталогическое проектирование 17](#_Toc150548267)

[4.2 Создание таблиц, индексов 20](#_Toc150548268)

[4.2.1 Создание таблиц 20](#_Toc150548269)

[4.2.2 Создание индексов 21](#_Toc150548270)

[4.3 Проектирование запросов к базе данных 22](#_Toc150548271)

[4.3.1 Симметричное внутреннее соединение с условием (два запроса с условием отбора по внешнему ключу, два – по датам) 22](#_Toc150548272)

[4.3.2 Симметричное внутреннее соединение без условия 24](#_Toc150548273)

[4.3.3 Левое внешнее соединение 26](#_Toc150548274)

[4.3.4 Правое внешнее соединение 26](#_Toc150548275)

[4.3.5 Запрос на запросе по принципу левого соединения 27](#_Toc150548276)

[4.3.6 Итоговый запрос без условия 28](#_Toc150548277)

[4.3.7 Итоговый запрос без условия с итоговыми данными вида: «всего», «в том числе» 28](#_Toc150548278)

[4.3.8 Итоговые запросы с условием на данные (по значению, по маске, с индексом, без индекса) 29](#_Toc150548279)

[4.3.9 Итоговый запрос с условием на группы 31](#_Toc150548280)

[4.3.10 Итоговый запрос с условием на данные и группы 32](#_Toc150548281)

[4.3.11 Запрос на запросе по принципу итогового запроса 33](#_Toc150548282)

[4.3.12 Запрос с использованием объединения 34](#_Toc150548283)

[4.3.13 Запросы с подзапросами (с использованием in, not in, case, операциями над итоговыми данными) 34](#_Toc150548284)

[4.3.14 Запросы из курсового проекта 37](#_Toc150548285)

[4.4 Создание представлений 39](#_Toc150548286)

[5 РАЗРАБОТКА КЛИЕНТСКОГО ПРИЛОЖЕНИЯ 41](#_Toc150548287)

[5.1 Формы и компоненты для работы с основными таблицами 41](#_Toc150548288)

[5.2 Формы и компоненты для работы со справочниками 44](#_Toc150548289)

[5.3 Формы и компоненты для отображения результатов запросов 47](#_Toc150548290)

[6 ТЕСТИРОВАНИЕ РАЗРАБОТАННОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ 49](#_Toc150548291)

[ЗАКЛЮЧЕНИЕ 50](#_Toc150548292)

[СПИСОК ЛИТЕРАТУРЫ 51](#_Toc150548293)

[ПРИЛОЖЕНИЕ А. ЛИСТИНГ СЕРВЕРНОГО ПРИЛОЖЕНИЯ 52](#_Toc150548294)

[ПРИЛОЖЕНИЕ Б. ЛИСТИНГ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ 69](#_Toc150548295)

[ПРИЛОЖЕНИЕ В. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 70](#_Toc150548296)

ВВЕДЕНИЕ

База данных — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. База данных обычно управляется системой управления базами данных (СУБД).

Данные в базах данных обычно хранятся в виде строк и столбцов, формирующих таблицу. Этими данными можно легко управлять, изменять, обновлять, контролировать и упорядочивать. В большинстве баз данных для записи и запросов данных используется язык структурированных запросов (SQL). SQL — это язык программирования, используемый в большинстве реляционных баз данных для запросов, обработки и определения данных, а также контроля доступа. SQL был разработан в 1970-х годах[1].

Основная задача курсового проекта – разработать реляционную базу данных «Ботанические сады» и пользовательское приложение для работы с составленной базой данных. В ходе разработки решается еще одна задача курсового проекта – закрепление практических навыков разработки и проектирования баз данных.

Приложение позволяет пользователю добавлять данные в таблицу, редактировать данные и удалять данные из таблиц. Также пользователю доступны функции поиск и фильтр для быстрого доступа к требующимся данным, генерация записей в таблицах и сохранение результатов некоторых запросов в форме отчета.

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ, ПОСТАНОВКА ЗАДАЧИ

В ходе курсового проекта необходимо разработать БД учета деятельности ботанических садов, а также приложение для работы с разработанной базой данных. Несколько садов может находится в одном городе. В садах работают рабочие. Сотрудники привязаны к саду и могут ухаживать за любым растением, растущим в этом саду. В садах растут только те растения, которые находятся в каталоге. Каждое растение относится к определённому виду, а вид – семейству.

Ведение дел с помощью компьютерного приложения имеет множество достоинств в сравнении с ведением дел с помощью бумажной отчетности.

Назначением программы является упрощение работы сотрудника ботанического сада вследствие ускорения получения и редактирования данных.

Функциональность разрабатываемой системы:

* просмотр, добавление, редактирование, удаление записей;
* поиск по конкретному значению;
* генерация диаграмм для некоторых итоговых запросов;
* генерация отчётов (в Excel виде) для запросов;

База данных содержит информацию о ботанических садах, растениях их видах и их семейств, удобрениях, сотрудниках, а также о посаженных растениях. Программа разрабатывается для сотрудников ботанических садов.

1.1 Описание входных документов

Входными документами являются информация о садах, сотрудниках, удобрениях и каталог растений.

Информация о садах строится из названия сада, города, года открытия, телефона, внешнего финансирования и типа собственности. Несколько садов может находится в одном городе. В садах растут растения. Сотрудники привязаны к саду и могут ухаживать за любым растением, растущим в этом саду. В садах растут только те растения, которые находятся в каталоге. Каждое растение относится к определённому виду, а вид – семейству. В таблице ухода за растениями описывается, когда какой сотрудник на каком растение использовал какое удобрением в каком количестве.

1.2 Описание выходных документов

Выходными документами ПО являются диаграммы по статистическим данным, а также отчёты по запросу.

Отчёты формируют MSOffice Excel, в которых отображается вся таблица результата запроса.

1.3 Список ограничений

Список ограничений, которые предусмотрены в базе данных:

* количество, использованного удобрения, должно быть строго больше 0 и не более 100;
* стаж сотрудника должен быть строго больше 0 и меньше 40;
* опыт сотрудника должен быть строго больше 0 и меньше 1000000;
* стоимость удобрения должна быть строго больше 0 и меньше 12000;
* год открытия сада должен быть строго больше 1970 и нестрого меньше текущего год;
* стоимость саженца должна быть строго больше 0 и меньше 10000;
* дата гибели растения должна быть позже даты посадки и раньше сегодняшней даты;
* дата посадки растения не может быть позже сегодняшней даты и не может раньше;

2 ОБОСНОВАНИЕ ВЫБОРА СУБД POSTGRESQL, ОПИСАНИЕ ВОЗМОЖНОСТЕЙ СУБД POSTGRESQL

Система управления базами данных (СУБД) - совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Были выделены такие наиболее популярные реляционные СУБД, как Oracle, MySQL, Microsoft SQL Server, PostgreSQL. Их достоинства заключаются в простой структуре, использовании широко распространённого SQL, быстром обновлении данных.

Для разработки приложения учета деятельности ботанических садов была выбрана реляционная СУБД PostgreSQL и графическая оболочка pgAdmin4v6, позволяющая создавать базы данных и таблицы с помощью встроенного конструктора с возможностью дальнейшего редактирования SQL-скриптов. Выбор PostgreSQL обосновывается удобством, простотой и широкой функциональностью данной СУБД, а также наличием графической оболочки pgAdmin4v6.

Возможности СУБД PostgreSQL[2] обширны. Данная СУБД надежна, устойчива, кроссплатформенна. Она использует многоверсионность для обеспечения надежной и быстрой работы в конкурентных условиях под высокой нагрузкой. PostgreSQL отлично использует современную архитектуру многоядерных процессоров. Расширяемость СУБД PostgreSQL поддерживает многочисленные типы данных. Система управления базами данных также позволяет создавать новые типы данных и методы доступа.

3 ОБОСНОВАНИЕ ВЫБОРА ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ ДЛЯ НАПИСАНИЯ КЛИЕНТСКОЙ ЧАСТИ, ПРОЕКТИРОВАНИЕ СТРУКТУРЫ ПО

Для написания клиентской части приложения выбран язык программирования C#[3] и система построения графических приложений WPF. Данный выбор обосновывается широкой функциональностью данного языка программирования, наличием библиотек для связи клиентского приложения с БД. Связь клиентской части приложения и базы данных реализована с помощью платформы FastAPI[4].

Приложение будет состоять из множества форм. На главной форме пользователю доступен просмотр таблиц, выбор действий над таблицами, просмотр запросов, генерация данных и очистка всех таблиц. Пользователь может переключаться между таблицами с помощью вкладок (TabControl).

Дополнительные формы открываются при нажатии на соответствующие кнопки. Разработаны формы для добавления, удаления, редактирования и поиска.

3.1 Не визуальные компоненты

В программе присутствуют несколько видов не визуальных компонентов. Одним из таких компонентов является SaveFileDialog, который используется для выбора расположения для сохранения MS Office Excel отчёта выполнения запросов.

Приложение может показывать диалоговые окна, например окна о предупреждении удаления элемента из таблицы или окна, информирующие об ошибке, это становится возможно благодаря компоненту MessageBox.

3.2 Визуальные компоненты отображения данных

Визуальные компоненты появляются на форме при создании компонента. Пользователь может напрямую взаимодействовать с данным типом компонентов.

В клиентском приложении использовалось множество визуальных компонентов. Основными компонентами отображения данных являются tabContol и dataGridView.

С помощью dataGridView пользователь может просмотреть результат выполнения запроса в виде таблицы. Данные результата запроса помещаются в свойство DataSource компонента dataGridView. С помощью кнопок пользователю доступно изменение данных, удаление строк и добавление строк в таблицу.

label используется для отображения количества строк в таблице на текущий момент. Для ввода данных используется textBox.

При вводе новых записей или редактировании существующих записей значение для поля с ссылкой на другую таблицу выбирается с помощью компонента comboBox.

4 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ В ВЫБРАННОЙ СУБД

Проектирование базы данных (БД) является одной из наиболее сложных и ответственных задач. Процесс проектирования БД представляет собой процесс переходов от неформального словесного описания информационной структуры предметной области к формализованному описанию объектов предметной области в терминах некоторой модели. Конечной целью проектирования является построение конкретной БД.

4.1 Проектирование концептуальной модели БД

Проектирование концептуальной модели БД состоит из нескольких основных этапов: инфологическое проектирование, даталогическое проектирование, физическое проектирование.

4.1.1 Инфологическое проектирование

На основании результатов анализа предметной области можно провести инфологическое проектирование информационной системы. В результате инфологического проектирования получим семантическую модель предметной области. Для этого необходимо выделить сущности, свойства сущностей и связи между сущностями.

Рассмотрим одну из сущностей предметной области – «Сад». Среди свойств этой сущности выделяются название, город, год открытия, номер телефона, финансирование, тип собственности.

Сущность «Сотрудник» имеет поля фамилия, имя, отчество, дата рождения, сад, за которым прикреплён сотрудник, должность, стаж работы и оклад.

Сущность «Растение» имеет поля сад, в котором растёт растение, название из каталога, дата посадки и гибели, стоимость саженца, условия ухода, фото и номер участка.

Таким образом, в результате инфологического проектирования была получена семантическая модель предметной области. В ней были выделены сущности модели данных, их свойства.

4.1.2 Даталогическое проектирование

В процессе даталогического проектирования выделяются таблицы и отношения между ними на основании построенной ранее семантической модели, схема данных приводится к третьей нормальной форме.

На рисунке 4.1 изображена реляционная модель данных для предметной области «Ботанические сады».

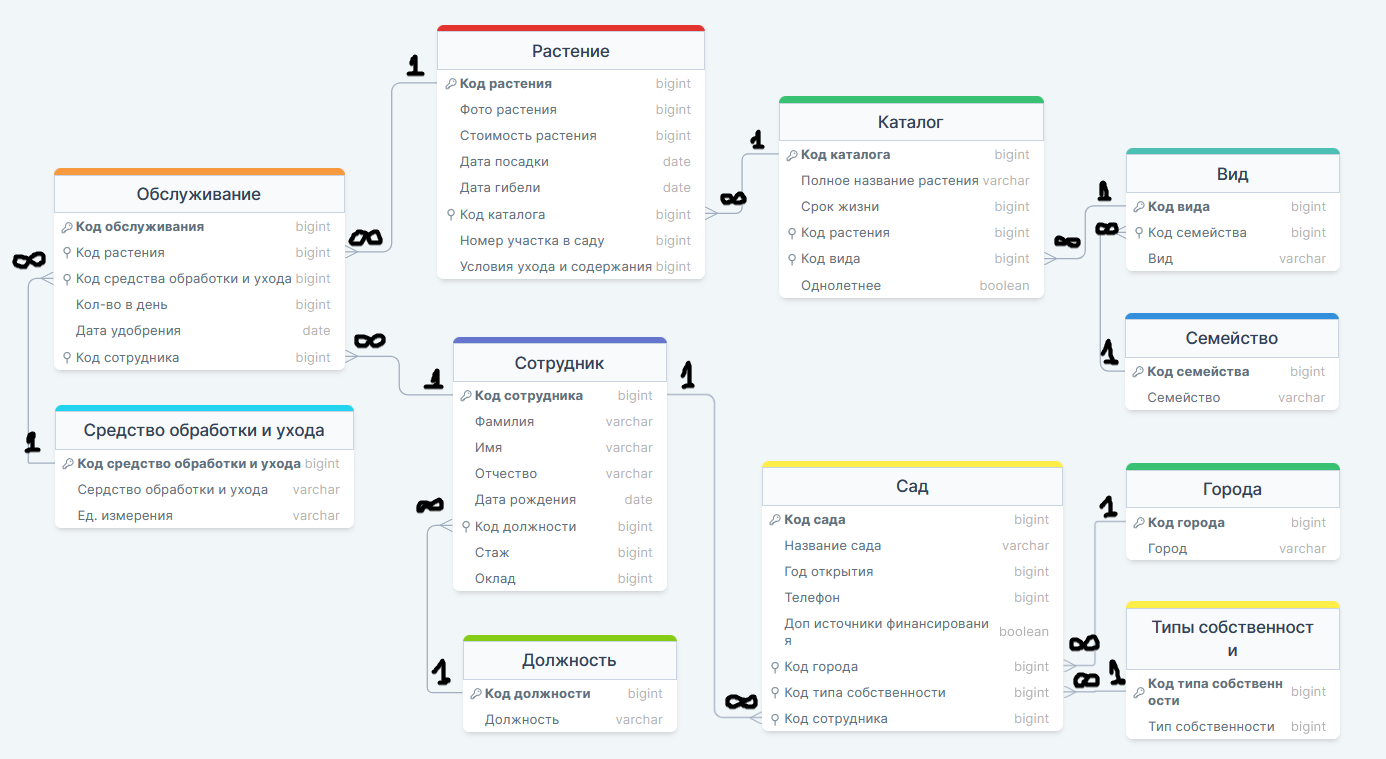


Рисунок 4.1 – Реляционная модель БД

Основными таблицами в модели данных являются таблицы:

* Сад;
* Сотрудник;
* Растение;
* Обслуживание;
* Каталог;
* Вид;

Справочными таблицами являются таблицы:

* Города;
* Семейства;
* Средства обработки и ухода
* Должности;
* Типы собственности;

4.1.3 Физическое проектирование

На этапе физического проектирования необходимо выделить общую структуру базы данных и определить, какие типы данных будут использованы для полей таблиц. Для упрощения восприятия выбранные типы и общая структура представлены в виде таблицы 4.1.

Таблица 4.1 – Таблицы и их поля

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица | Поле | Назначение | Тип данных |
| care\_products | id | Код средства ухода | integer |
| care\_product | Название | charactervarying |
| unit | Единица измерения | charactervarying |
| catalogue | id | Код каталога | integer |
| full\_name | Полное название | charactervarying |
| life\_span | Срок жизни | integer |
| genera\_id | Код вида | integer |
| annual | Однолетнее | boolean |
| cities | id | Код города | integer |
| city | Название | charactervarying |
| employees | id | Код работника | integer |
| surname | Фамилия | charactervarying |
| name | Имя | charactervarying |
| patronymic | Отчество | charactervarying |
| birth | Дата рождения | date |
| position\_id | Код должности | integer |
| experience | Опыт работы | integer |
| salary | Зарплата | integer |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| families | id | Код семейства | integer |
| family | Название | charactervarying |
| gardens | id | Код сада | integer |
| garden | Название | charactervarying |
| opening | Год открытия | integer |
| phone | Телефон | bigint |
| financing | Финансироване | boolean |
| city\_id | Код города | integer |
| property\_id | Код типа Собственности | integer |
| employee\_id | Код работника | integer |
| generas | id | Код вида | integer |
| family\_id | Код семейства | integer |
| genera | Название | charactervarying |
| plants | id | Код растения | integer |
| photo | Фото | bytea |
| price | Цена | integer |
| planting | Дата посадки | date |
| death | Дата гибели | date |
| catalogue\_id | Код каталога | integer |
| plot | Номер участка | integer |
| conditions | Условия содержания | text |
| positions | id | Код должности | integer |
| position | Название | charactervarying |
| properties | id | Код типа собственности | integer |
| property | Название | charactervarying |
| service | id | Код обслуживания | integer |
| plant\_id | Код растения | integer |
| care\_product\_id | Код средсва ухода | integer |
| amount | Кол-во | integer |
| fertilized | Дата удобрения | date |
| employee\_id | Код работника | integer |

4.2 Создание таблиц, индексов

На рисунке 4.2 представлен скрипт создания базы данных для курсового проекта.

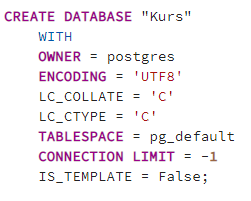


Рисунок 4.2 – Скрипт создания БД

4.2.1 Создание таблиц

Таблицы для базы данных были созданы при помощи SQL-запросов. На рисунках 4.3 - 4.5 представлены скрипты создания некоторых таблиц.

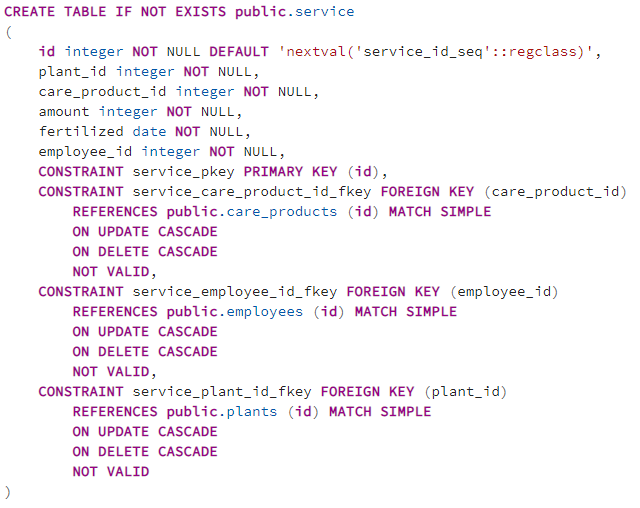


Рисунок 4.3 – Скрипт создания таблицы «Обслуживание»

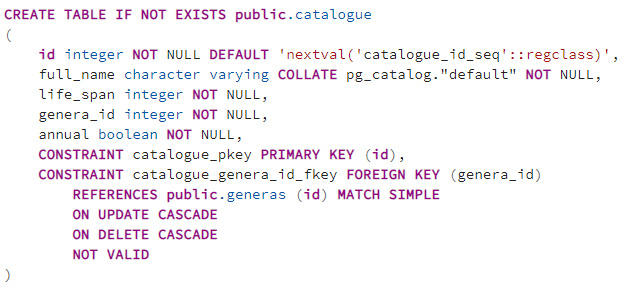


Рисунок 4.4 – Скрипт создания таблицы «Каталог»

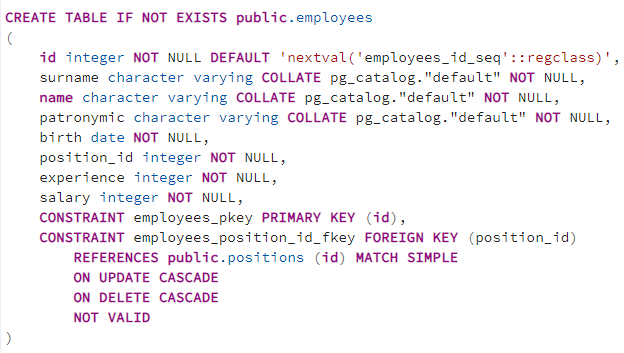


Рисунок 4.5 – Скрипт создания таблицы «Сотрудник»

4.2.2 Создание индексов

Индекс - объект базы данных, создаваемый с целью повышения производительности поиска данных.

На рисунке 4.6 изображены скрипты создания индекса для таблицы «Обслуживание».

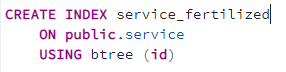


Рисунок 4.6 –service\_fertilized для таблицы «Обслуживание»

4.3 Проектирование запросов к базе данных

Запросы играют важнейшую роль в работе с базой данных. Запрос – это обращение к базе данных для получения требуемых данных и выполнения действий с ними.

В курсовом проекте требуется реализовать различные виды запросов: итоговые, запросы с подзапросами и т.п.

4.3.1 Симметричное внутреннее соединение с условием (два запроса с условием отбора по внешнему ключу, два – по датам)

Запрос внутреннего соединение с условием по внешнему ключу**.** Все сады с типом собственности "Смешанная собственность" (см. рис 4.7-4.8)

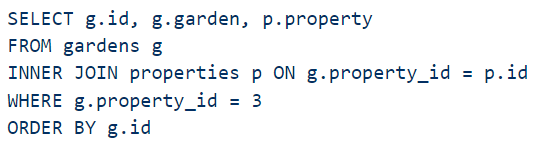


Рисунок 4.7 – Код запроса

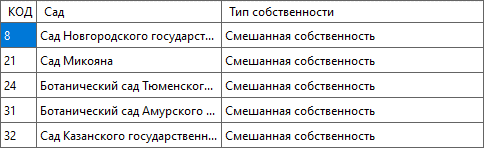


Рисунок 4.8 – Результат запроса

Запрос внутреннего соединение с условием по внешнему ключу. Все сады в городе "Санкт-Петербург" (см. рис 4.9-4.10)

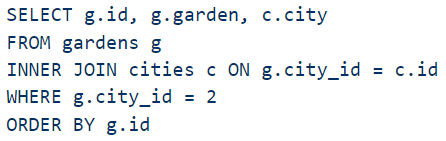


Рисунок 4.9 –– Код запроса

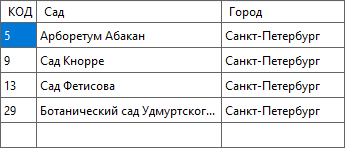


Рисунок 4.10 – Результат запроса

Запрос внутреннего соединение с условием даты. Список ухода за последнюю неделю (см. рис 4.11-4.12)

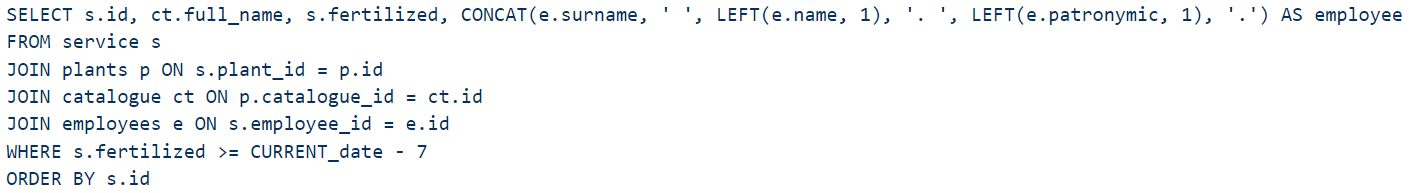


Рисунок 4.11 – Код запроса

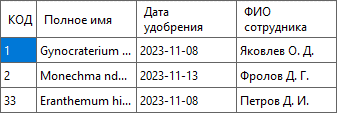


Рисунок 4.12 – Результат запроса

Запрос внутреннего соединение с условием даты. Список ухода за последний сезон (см. рис. 4.13-4.14)

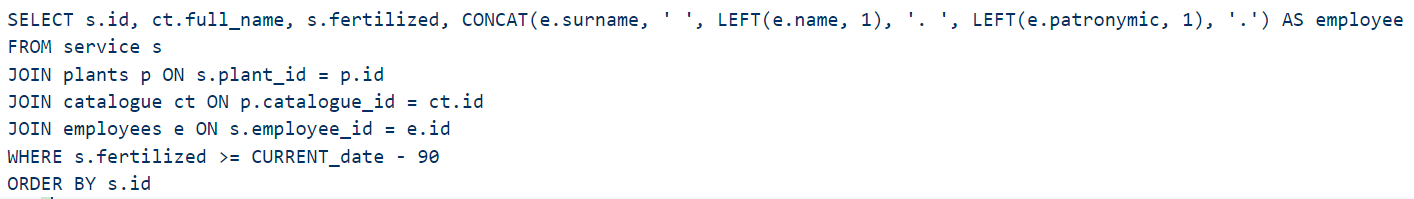


Рисунок 4.13 – Код запроса



Рисунок 4.14 – Результат запроса

4.3.2 Симметричное внутреннее соединение без условия

Запрос внутреннего соединение без условия. Вывести работника и его должность (см. рис. 4.15-4.16)

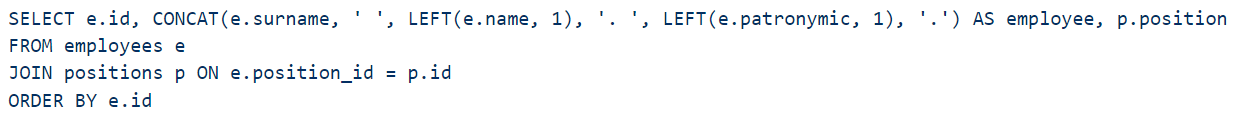


Рисунок 4.15 – Код запроса

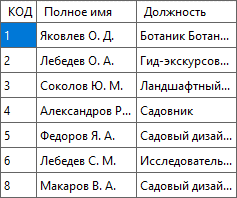


Рисунок 4.16 – Результат запроса

Запрос внутреннего соединение без условия. Вывести сад и его тип собственности (см. рис. 4.17-4.18)

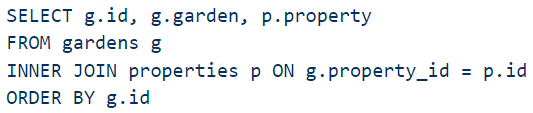


Рисунок 4.17 – Код запроса

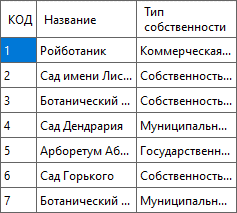


Рисунок 4.18 – Результат запроса

Запрос внутреннего соединение без условия. Вывод сада и его города (см. рис 4.19-4.20)

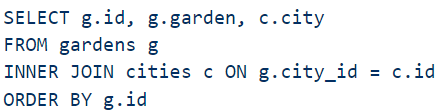


Рисунок 4.19 – Код запроса

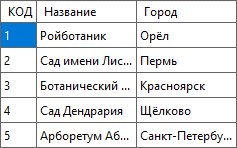


Рисунок 4.20 – Результат запроса

4.3.3 Левое внешнее соединение

Вывод сотрудников, участвующих в уходе (см. рис. 4.21–4.22)

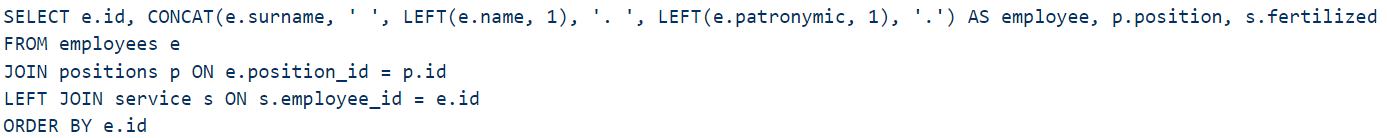


Рисунок 4.21 – Код запроса

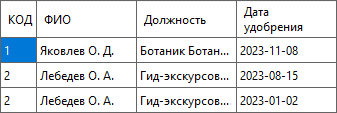


Рисунок 4.22 – Результат запроса

4.3.4 Правое внешнее соединение

Вывод сотрудников, ухаживающих за еще не умершими растениями (см. рис. 4.23 – 4.24)

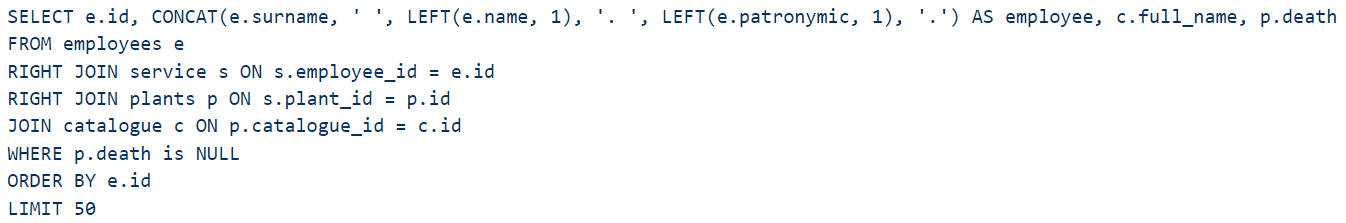


Рисунок 4.23 – Код запроса

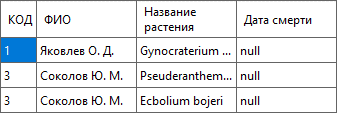


Рисунок 4.24 – Результат запроса

4.3.5 Запрос на запросе по принципу левого соединения

Вывод информации о количестве использованного удобрения за сезон (см. рис. 4.25 – 4.26)

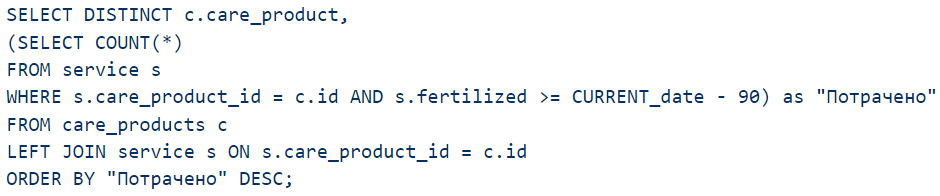


Рисунок 4.25 – Код запроса

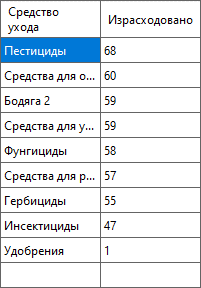


Рисунок 4.26 – Результат запроса

4.3.6 Итоговый запрос без условия

Вывод количества видов в семействах (см. рис. 4.27 – 4.28)

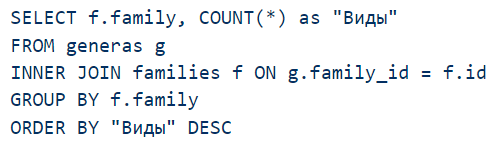


Рисунок 4.27 – Код запроса

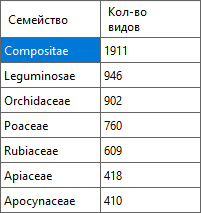


Рисунок 4.28 – Результат запроса

4.3.7 Итоговый запрос без условия с итоговыми данными вида: «всего», «в том числе»

Вывод информации о количестве садов с финансированием и без   
(см. рис. 4.29 – 4.30)

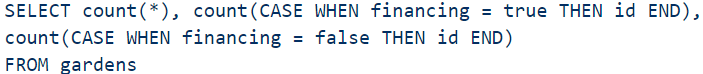


Рисунок 4.29 – Код запроса

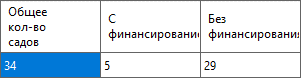


Рисунок 4.30 – Результат запроса

4.3.8 Итоговые запросы с условием на данные (по значению, по маске, с индексом, без индекса)

Итоговые запросы с условием на данные по значению. Вывод количества опр. типа собственности за период после 2010г (см. рис. 4.31– 4.32)

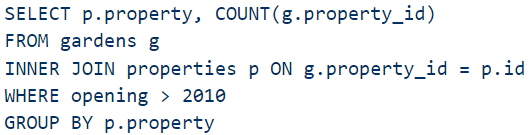


Рисунок 4.31 – Код запроса

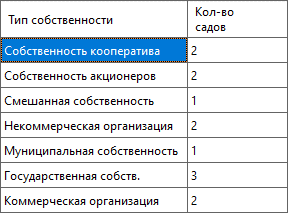


Рисунок 4.32 – Результат запроса

Итоговые запросы с условием на данные по маске. Вывод количества растений, обслуживаемых Владимирами. (см. рис. 4.33 – 4.34)

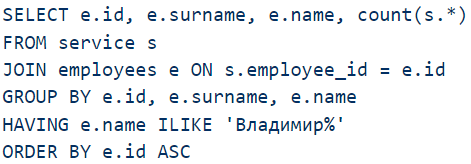


Рисунок 4.33 – Код запроса

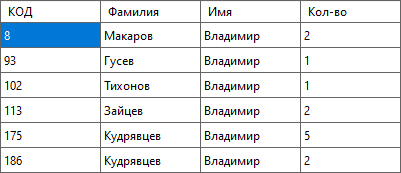


Рисунок 4.34 – Результат запроса

Итоговые запросы с условием на данные с использованием индекса. Получить количество садов в каждом городе и указать максимальное открытие среди них. (см. рис. 4.35 – 4.36)

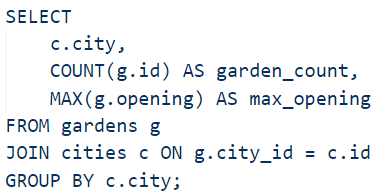


Рисунок 4.35 – Код запроса

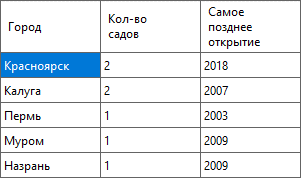


Рисунок 4.36 – Результат запроса

Итоговые запросы с условием на данные без использования индекса. Вывести количество растений определенного вида. (см. рис. 4.42 – 4.43)

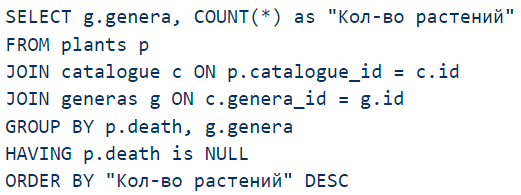


Рисунок 4.37 – Код запроса

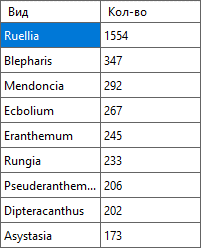


Рисунок 4.38 – Результат запроса

4.3.9 Итоговый запрос с условием на группы

Вывод количество растений определенного вида, которые погибли (см. рис. 4.39 – 4.40)

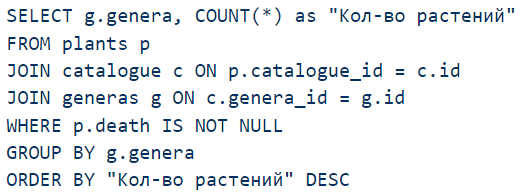


Рисунок 4.39 – Код запроса

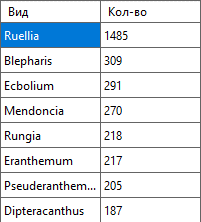


Рисунок 4.40 – Результат запроса

4.3.10 Итоговый запрос с условием на данные и группы

Вывод растения, где средняя стоимость саженцев не превышает 888 (см. рис. 4.41 – 4.42)

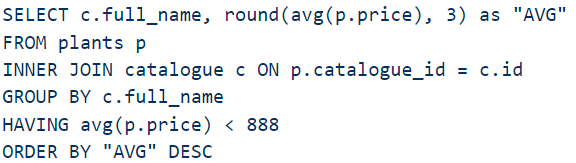


Рисунок 4.41 – Код запроса

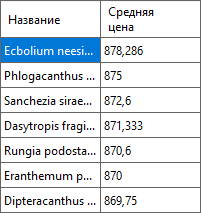


Рисунок 4.42 – Результат запроса

4.3.11 Запрос на запросе по принципу итогового запроса

Информация о rоличествe растений удобренных за 90 дней, превышающих 2. (см. рис 4.43 – 4.44)

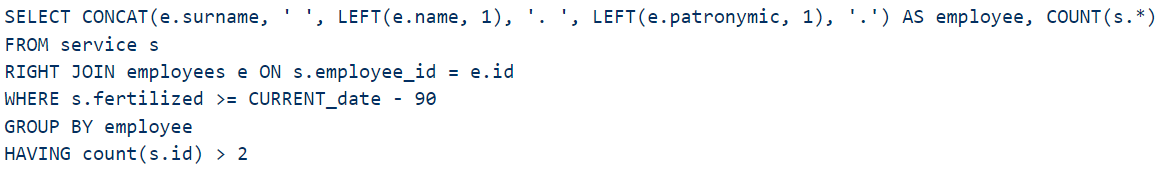


Рисунок 4.43 – Код запроса

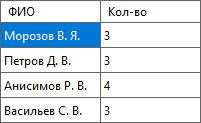


Рисунок 4.44 – Результат запроса

4.3.12 Запрос с использованием объединения

Вывод количества политых растениях каждым сотрудником (см. рис. 4.45 – 4.46)

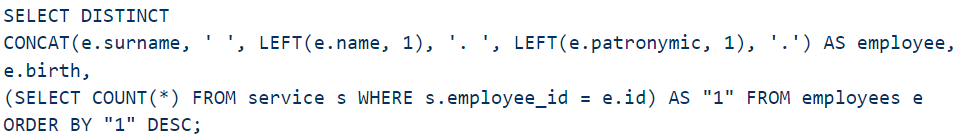


Рисунок 4.45– Код запроса

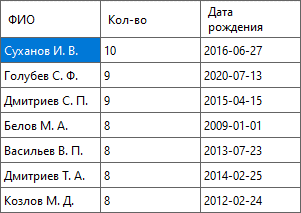


Рисунок 4.46 – Результат запроса

4.3.13 Запросы с подзапросами (с использованием in, not in, case, операциями над итоговыми данными)

Запрос с подзапросом с использованием in. Вывести работников с опытом работы более 2 лет и их обслуживании (см. рис 4.47 – 4.48)

Запрос с подзапросом с использованием not in. Вывести дату удобрения растения с сроком жизни больше 5 лет (см. рис 4.49 – 4.50)

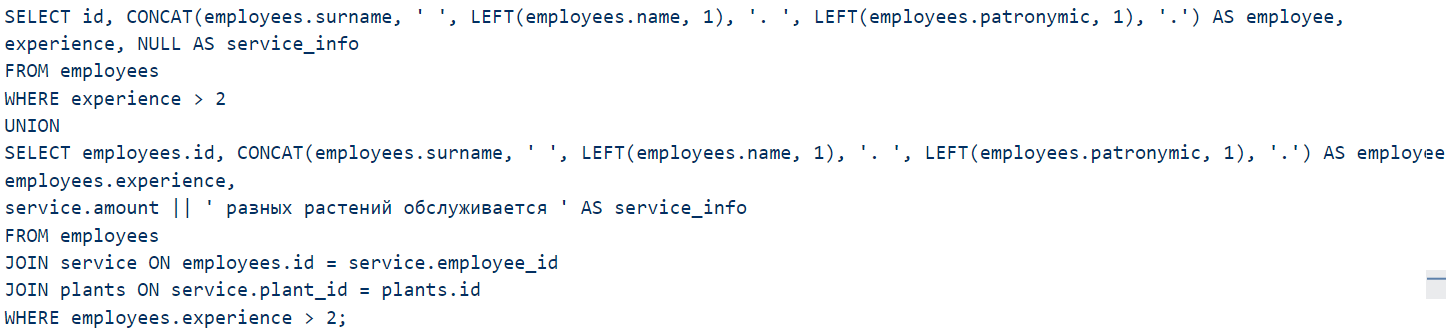


Рисунок 4.47 – Код запроса

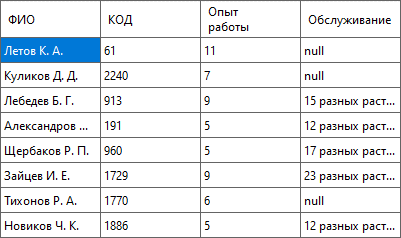


Рисунок 4.48 – Результат запроса

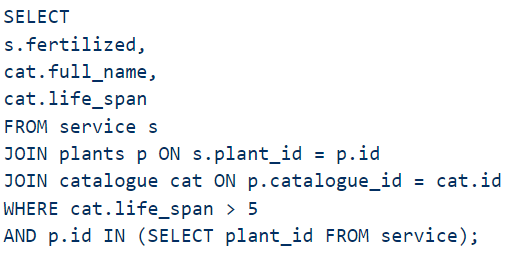


Рисунок 4.49 – Код запроса

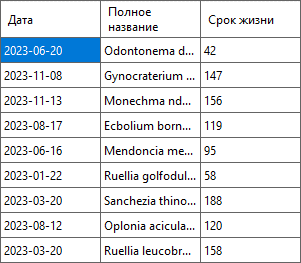


Рисунок 4.50 – Результат запроса

Запрос с подзапросом с использованием case. Вывести pастения без обслуживания и обслуживающих сотрудников. (см. рис 4.51 – 4.52)

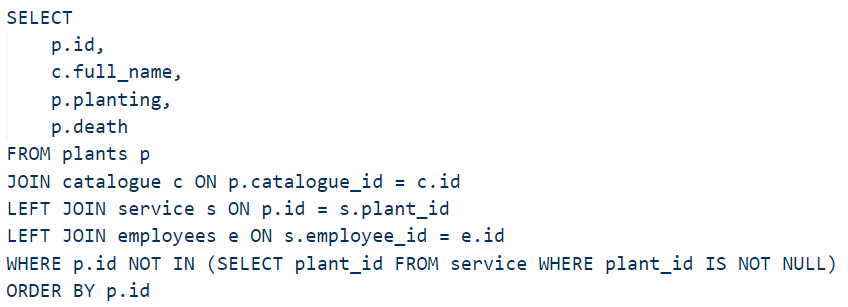


Рисунок 4.51 – Код запроса

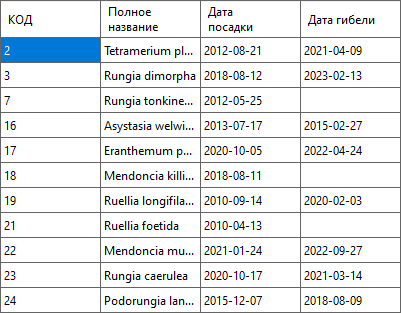


Рисунок 4.52 – Результат запроса

Запрос с подзапросом с операциями над итоговыми данными. Получить список сотрудников, их должности и указать "Молодой" или "Опытный" в зависимости от опыта работы. (см. рис 4.53 – 4.54)

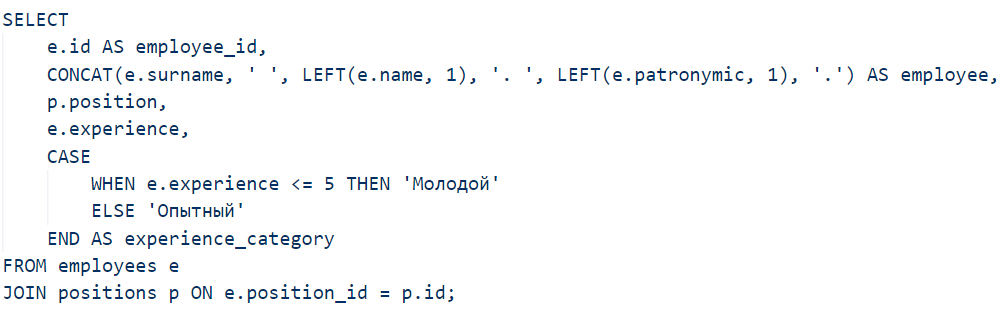


Рисунок 4.53 – Код запроса

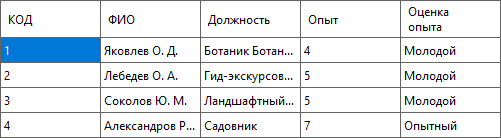


Рисунок 4.54 – Результат запроса

4.3.14 Запросы из курсового проекта

Определить процент растений, посаженных до 2010 года по каждому ботаническому саду и в целом по всем садам (см. рис. 4.55 – 4.56)

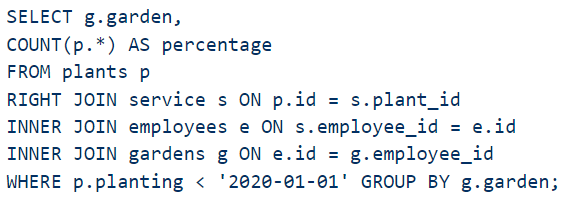


Рисунок 4.57 – Код запроса

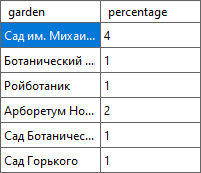


Рисунок 4.58 – Результат запроса

Определить процент растений, посаженных до 2020 года по каждому ботаническому саду и в целом по всем садам (см. рис. 4.59– 4.60)

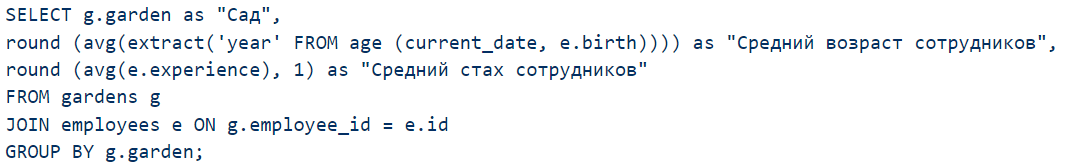


Рисунок 4.59 – Код запроса

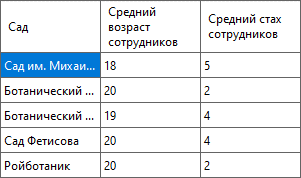


Рисунок 4.60 – Результат запроса

Определить суммарные расходы по уходу за растениями и количество растений по каждому саду (см. рис. 4.61 – 4.62)

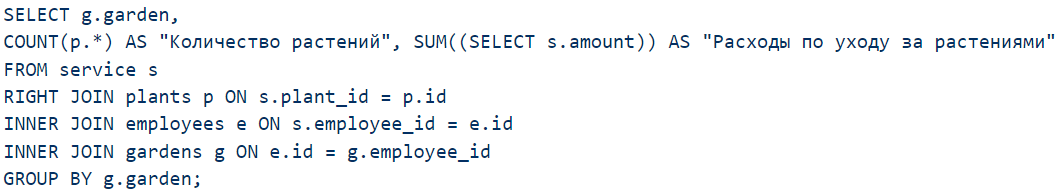


Рисунок 4.61 – Код запроса

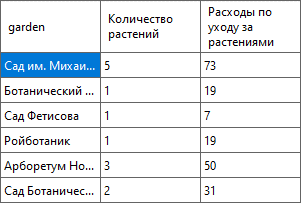


Рисунок 4.62 – Результат запроса

4.4 Создание представлений

Представления – это способ задать имя для сложного запроса SELECT на выборку данных. На рисунках 4.63 – 4.64 изображены SQL-код созданий представлений.

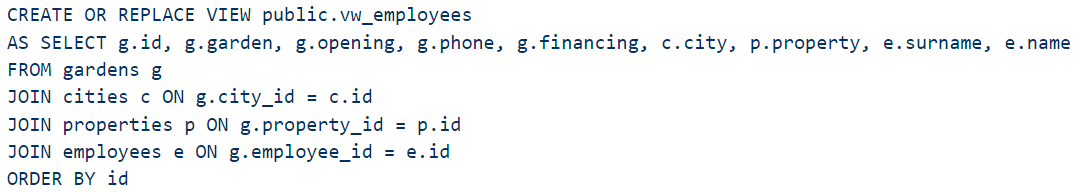


Рисунок 4.63 – Представление «vw\_employees»

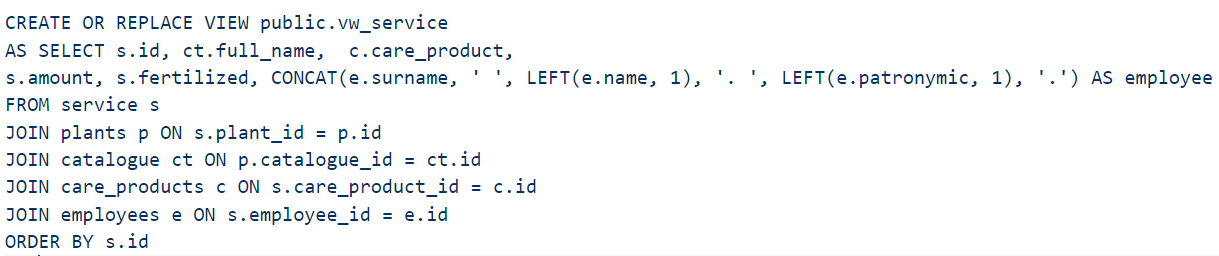


Рисунок 4.64 – Представление «vw\_service»

5 РАЗРАБОТКА КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

Клиентское приложение разрабатывается, как Windows Forms приложение на языке программирования C#.

Приложение состоит из главной формы и множества дополнительных форм для работы с БД. На главной форме есть 12 кнопок для работы с таблицами, справочниками и запросами.

На каждой странице (для всех таблиц) есть кнопки «Добавить», «Редактировать», «Удалить», а также реализован поиск по одной из колонок.

Для начала работы с приложением нужно совершить авторизацию. На рисунке 5.1 изображена форма авторизации

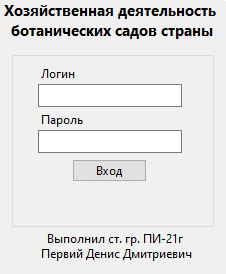


Рисунок 5.1 – Форма авторизации

5.1 Формы и компоненты для работы с основными таблицами

При выборе таблицы в dataGridView загружается содержимое таблицы из БД. На рисунках 5.2 – 5.4 изображены страницы и компоненты для работы с некоторыми основными таблицами.

На рисунках 5.5 – 5.6 изображены некоторые формы для добавления и редактирования записей в таблицах.

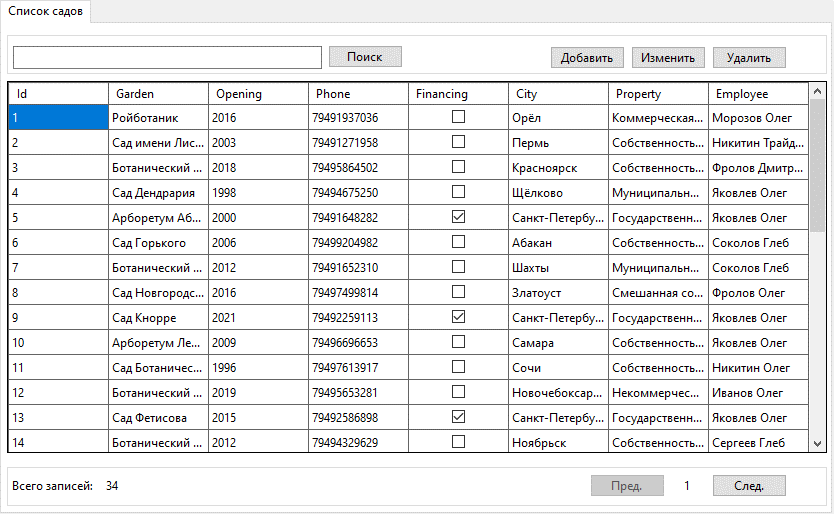


Рисунок 5.2 – Таблица «Сады»

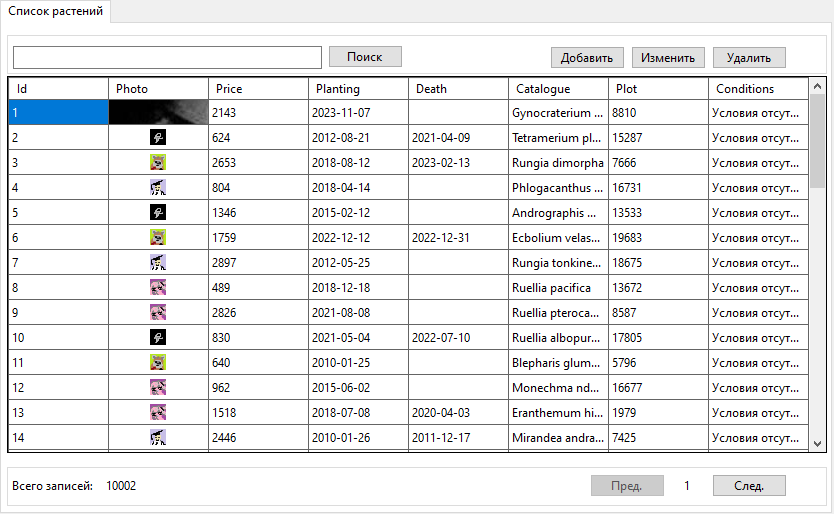


Рисунок 5.3 – Таблица «Растения»

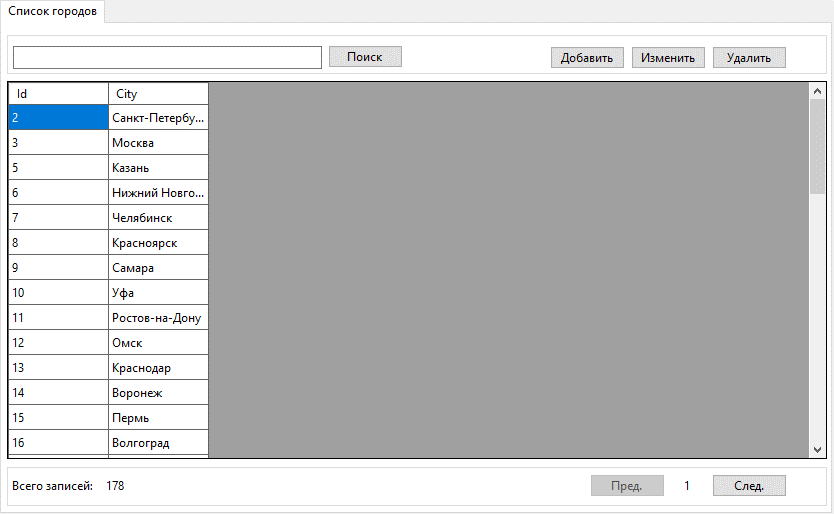


Рисунок 5.4 – Таблица «Города»

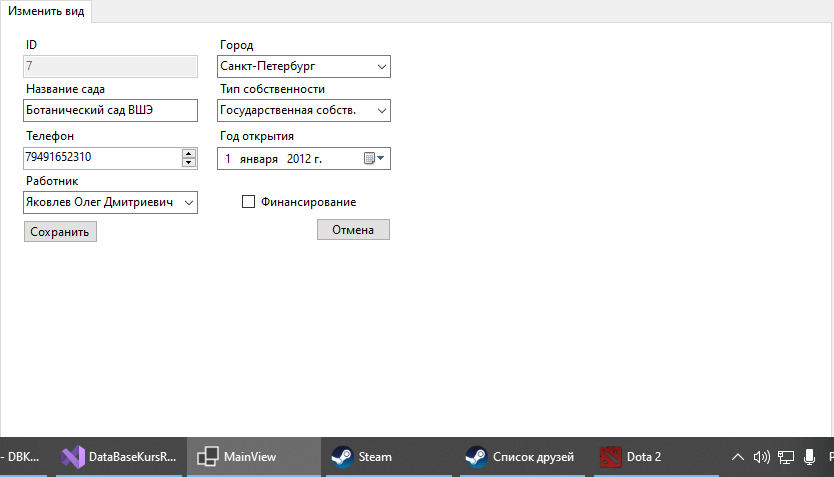


Рисунок 5.5 – Форма добавления и редактирования сада

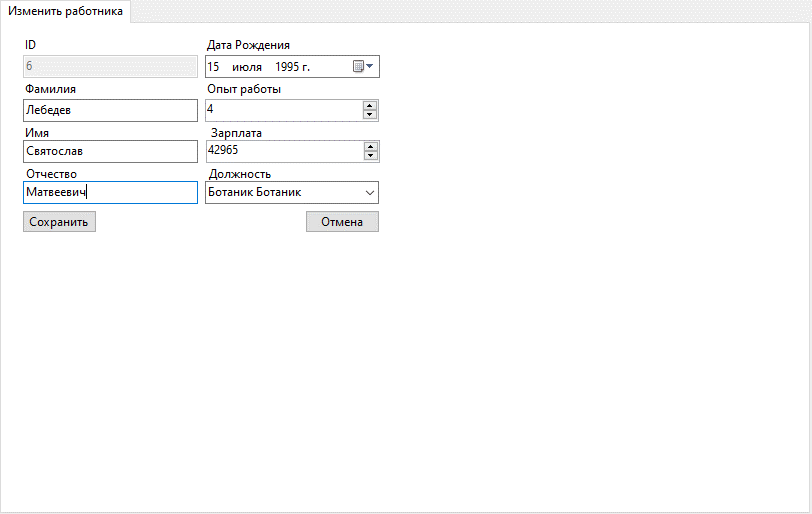


Рисунок 5.6 – Форма добавления и редактирования сотрудника

5.2 Формы и компоненты для работы со справочниками

На рисунках 5.7 – 5.9 изображены страницы работы со справочниками.

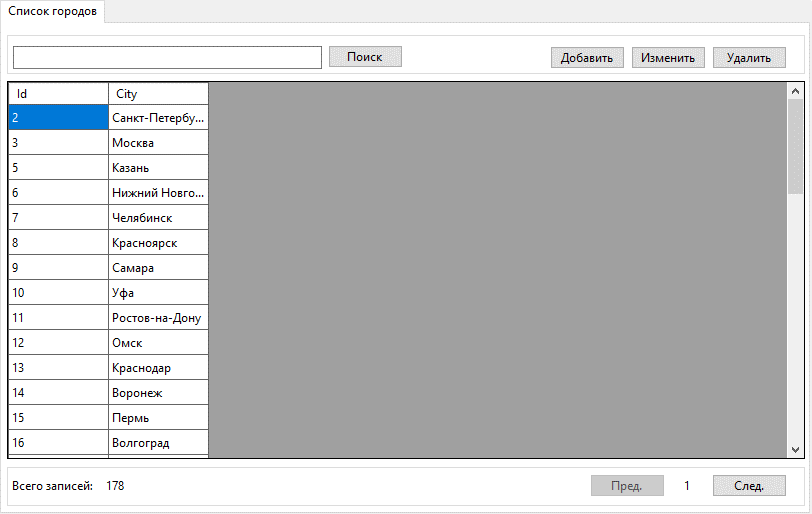


Рисунок 5.7 – Справочник «Города»

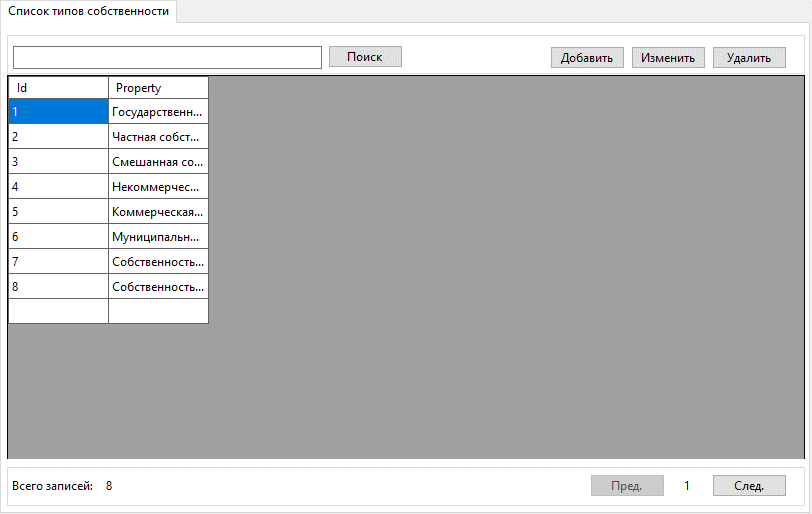


Рисунок 5.8 – Справочник «Типы собственности»

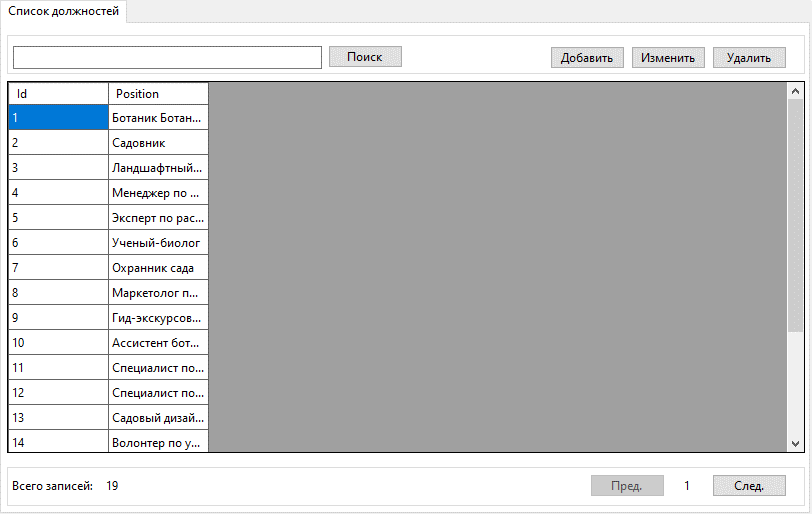


Рисунок 5.9 – Справочник «Должности»

На рисунках 5.10 – 5.12 изображены формы для добавления и редактирования записей в справочниках.

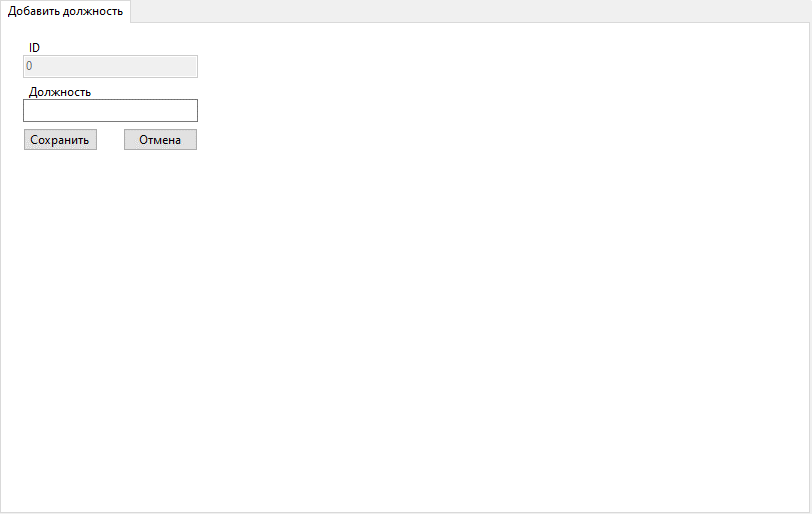


Рисунок 5.10 – Форма добавления должности

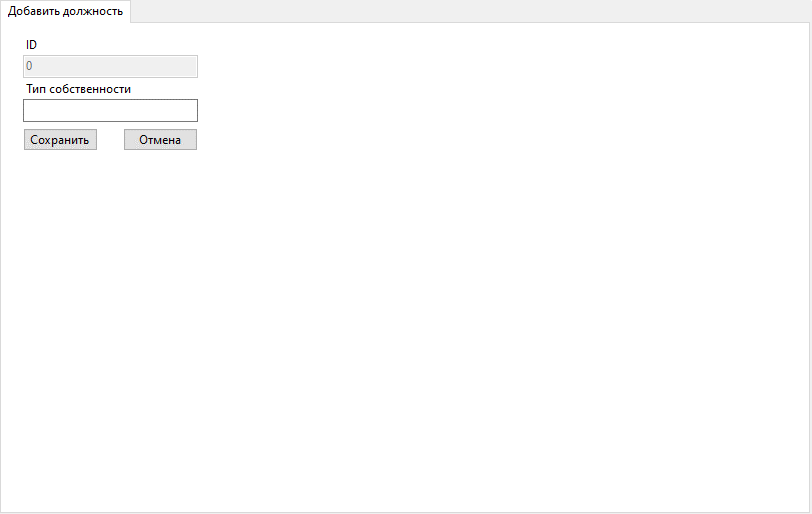


Рисунок 5.11 – Добавление типа собственности

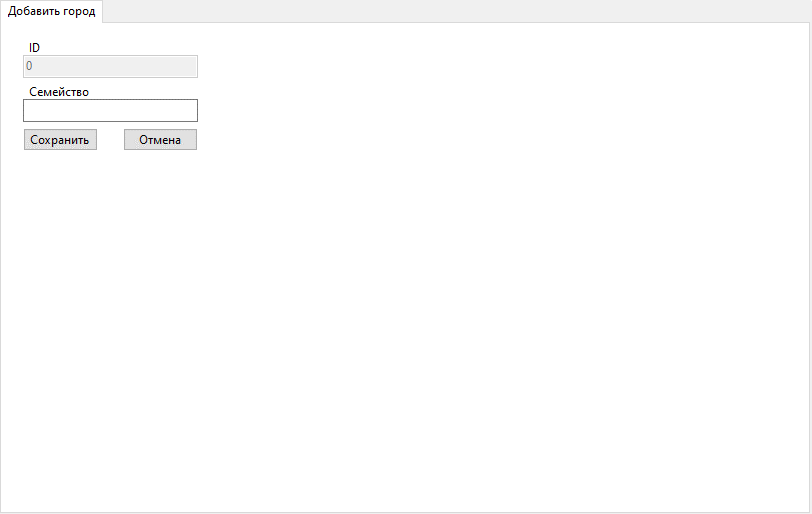


Рисунок 5.12 – Добавление семейства растений

5.3 Формы и компоненты для отображения результатов запросов

Для отображения результатов запросов создана отдельная страница. Результаты запросов выводятся в dataGridView. Выбрать запрос можно с помощью comboBox заполнен запросами. Также на странице запросов есть кнопка «Экспорт в Excel», а для некоторых итоговых запросов можно построить диаграмму. На рисунке 5.13 изображена страница запросов.

При нажатии на кнопку «Экспорт в Excel» открывается диалоговое окно, в котором пользователю предоставляется возможность выбрать место и название для файла Excel. Экспорт в Excel производится с помощью сторонней библиотеки EPPlus[5].

При нажатии на кнопку «Диаграмма» открывается окно с двумя видами диаграмм. (см. рис 5.16) Диаграммы построены и выведены на экран с помощью сторонней библиотеки OxyPlot[6].

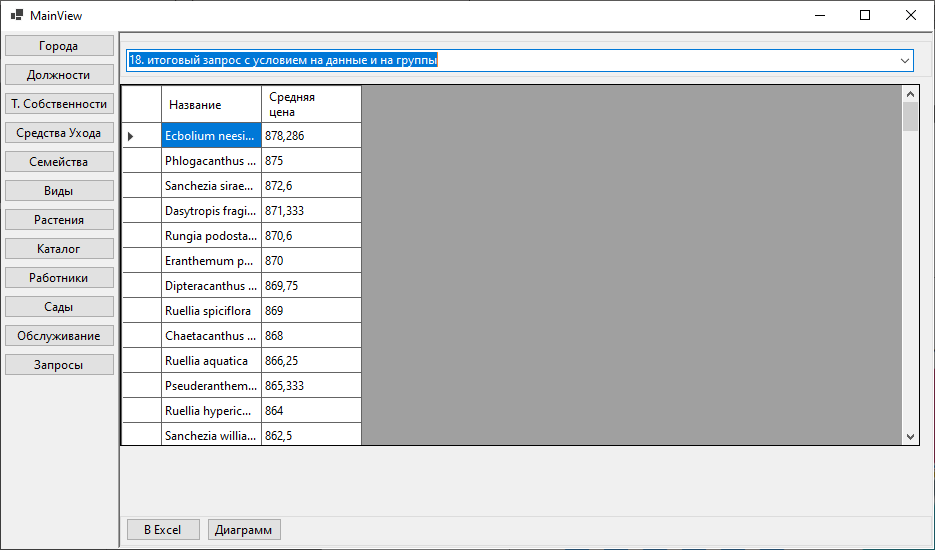


Рисунок 5.15 – Форма выбора запроса

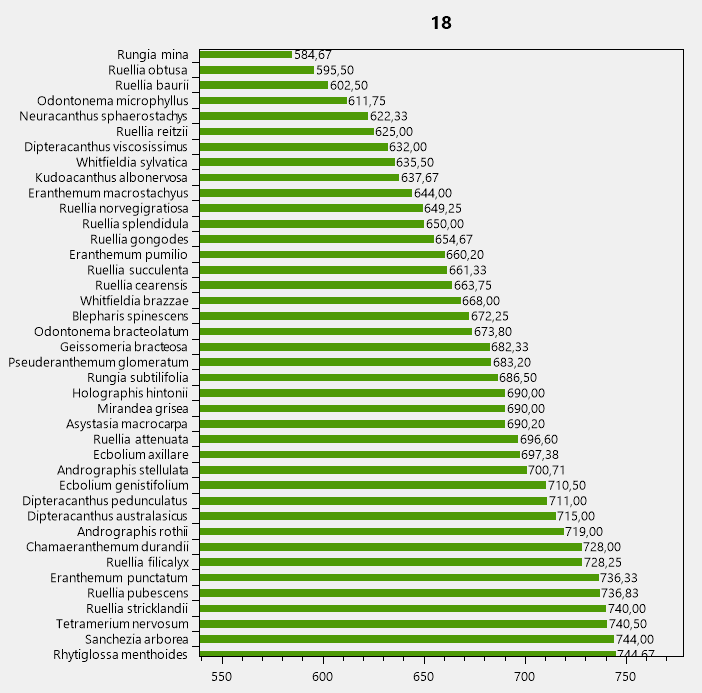


Рисунок 5.16 – Форма с диаграммами

6 ТЕСТИРОВАНИЕ РАЗРАБОТАННОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Для тестирования системы был разработан алгоритм генерации псевдослучайных записей для последующего добавления их в базу данных. Программа генерирует случайные наборы и добавляет их в базу данных.

Для корректной работы программы-генератора необходимо, чтобы таблицы-справочники базы данных были заполнены.

Для защиты от несанкционированного доступа перед запуском основной формы пользователь должен авторизоваться. Если пользователь ввел неверные данные, запрос на подключение отклоняется. На рисунке 5.1 изображена форма авторизации.

Также в приложении предусмотрена проверка корректности введенных данных для каждой формы добавления и редактирования данных.

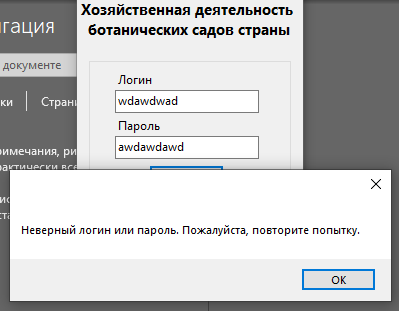


Рисунок 6.1 – Ошибка входа.

По результатам тестирования программы можно сказать, что она работоспособна. По завершении разработки и отладки программного продукта было написано краткое руководство пользователя, в котором описаны основные положения и рекомендации по использованию программы. Приложение пользователя приведено в приложении Г.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта все поставленные цели были успешно достигнуты. Основная задача проекта заключалась в разработке реляционной базы данных "Ботанические сады" и соответствующего пользовательского приложения для работы с данной базой данных, и эта цель была успешно реализована. Кроме того, в процессе разработки была решена еще одна задача проекта, связанная с закреплением практических навыков в области разработки и проектирования баз данных.

Было создано клиентское приложение, предназначенное для учета деятельности ботанических садов страны, а также создана и заполнена соответствующая база данных. Разработаны различные виды запросов и соответствующие диаграммы на основе результатов этих запросов. Реализована функциональность сохранения результатов запроса в файл Excel.

В процессе выполнения проекта были освоены язык структурированных запросов SQL и СУБД PostgreSQL, включая графическую оболочку pgAdmin. Также приобретены навыки программирования на C# и изучена платформа FastAPI, которая обеспечивает связь между базой данных и клиентским приложением.

Разработанное приложение предоставляет практическую полезность для ботанических садов страны, предоставляя им возможность эффективно управлять своей деятельностью. Для пользователей системы составлено соответствующее руководство, облегчающее использование приложения.

СПИСОК ЛИТЕРАТУРЫ

1. Oracle. Что такое база данных? – URL: <https://www.oracle.com/cis/database/what-is-database/> (дата обращений: 10.11.2023)
2. PostgreSQL. – URL: https://www.postgresql.org/ (дата обращения: 10.11.2023)
3. Metanit. C# Документация. – URL: https://metanit.com/sharp/ (дата обращения: 10.11.2023)
4. FastAPI – URL: https://fastapi.tiangolo.com/ (дата обращения: 10.11.2023)
5. EPPlus – URL: https://epplussoftware.com/ (дата обращения: 10.11.2023)
6. OxyPLot – URL: https://fastapi.tiangolo.com/ (дата обращения: 10.11.2023)

ПРИЛОЖЕНИЕ А.  
ЛИСТИНГ СЕРВЕРНОГО ПРИЛОЖЕНИЯ

import psycopg2

import base64

from fastapi import FastAPI, Response, status

from fastapi.middleware.cors import CORSMiddleware

from db\_models import \*

from queries import router as queries\_router, get\_cursor

app = FastAPI()

app.include\_router(queries\_router)

# `uvicorn main:app --reload`

# Разрешить доступ с источника React-приложения

origins = ["http://localhost:3000"]

app.add\_middleware(

    CORSMiddleware,

    allow\_origins=origins,

    allow\_credentials=True,

    allow\_methods=["\*"],

    allow\_headers=["\*"],

)

@app.get('/{table\_name}/getCols', status\_code=200)

def get\_table\_cols(table\_name: str, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(

                f"SELECT column\_name FROM information\_schema.columns WHERE table\_name = '{table\_name}'")

            column\_names = [col[0].replace("\_id", "")

                            for col in cursor.fetchall()]

            response.status\_code = status.HTTP\_200\_OK

        return {"column\_names": column\_names}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.get('/ref/{reference\_name}/getAll', status\_code=200)

def get\_all\_rows(reference\_name: str, page: int, limit: int, response: Response):

    offset = (page - 1) \* limit

    try:

        with get\_cursor() as cursor:

            cursor.execute(

                f"SELECT \* FROM {reference\_name} ORDER BY id LIMIT {limit} OFFSET {offset}")

            rows = cursor.fetchall()

            cursor.execute(

                f"SELECT column\_name FROM information\_schema.columns WHERE table\_name = '{reference\_name}'")

            cols = cursor.fetchall()

            result = []

            for row in rows:

                row\_dict = {}

                for i, col in enumerate(cols):

                    row\_dict[col[0]] = row[i]

                result.append(row\_dict)

            cursor.execute(f"SELECT COUNT(\*) FROM {reference\_name}")

            total\_rows = cursor.fetchone()[0]

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result, "total": total\_rows}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"An unhandled error occurred: {e}"}

@app.delete('/{table\_name}/delete/{table\_id}', status\_code=200)

def delete\_row(table\_name: str, table\_id: int, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"DELETE FROM {table\_name} WHERE id = {table\_id}")

            response.status\_code = status.HTTP\_200\_OK

            return {"success": f"Удаление успешно."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        print(e)

        return {"error": f"Возникла необработанная ошибка: {e}"}

# Cities

@app.post('/cities/add', status\_code=200)

def add\_city(data: Cities, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(

                f"INSERT INTO cities VALUES (DEFAULT, '{data.city}')")

            response.status\_code = status.HTTP\_201\_CREATED

            return {"success": f"Город c названием '{data.city}' добавлен."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.get('/cities/getByValue', status\_code=200)

def get\_city\_by\_value(city: str, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                            SELECT \* FROM cities

                            WHERE city ILIKE '%{city}%'

                            ORDER BY id

                            LIMIT 50

                            """)

            city = {

                "id": "",

                "city": "",

            }

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                city["id"] = row[0]

                city["city"] = row[1]

                result.append(dict(city))

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/cities/update/{city\_id}', status\_code=200)

def update\_city(city\_id: int, data: Cities, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(

                f"UPDATE cities SET city = '{data.city}' WHERE id = {city\_id}")

            response.status\_code = status.HTTP\_200\_OK

            return {"success": "Город успешно изменен."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

# Properties

@app.get('/properties/getByValue', status\_code=200)

def get\_position\_by\_value(position: str, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                            SELECT \* FROM properties

                            WHERE position ILIKE '%{position}%'

                            ORDER BY id

                            LIMIT 50

                            """)

            position = {

                "id": "",

                "position": "",

            }

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                position["id"] = row[0]

                position["position"] = row[1]

                result.append(dict(position))

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/properties/add', status\_code=200)

def add\_position(data: Properties, response: Response):

    try:

        with get\_cursor() as cursor:

            print(data)

            cursor.execute(

                f"INSERT INTO properties VALUES (DEFAULT, '{data.position}')")

            response.status\_code = status.HTTP\_201\_CREATED

            return {"success": "Тип собственности успешно добавлен."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/properties/update/{position\_id}', status\_code=200)

def update\_position(position\_id: int, data: Properties, response: Response):

    cursor = get\_cursor()

    try:

        with get\_cursor() as cursor:

            print(position\_id)

            cursor.execute(

                f"UPDATE properties SET position = '{data.position}' WHERE id = {position\_id}")

            response.status\_code = status.HTTP\_200\_OK

            return {"success": "Тип собственности успешно изменен."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

# Positions

@app.get('/positions/getByValue', status\_code=200)

def get\_position\_by\_value(position: str, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                            SELECT \* FROM positions

                            WHERE position ILIKE '%{position}%'

                            ORDER BY id

                            LIMIT 50

                            """)

            position = {

                "id": "",

                "position": "",

            }

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                position["id"] = row[0]

                position["position"] = row[1]

                result.append(dict(position))

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/positions/add', status\_code=200)

def add\_position(data: Positions, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(

                f"INSERT INTO positions VALUES (DEFAULT, '{data.position}')")

            response.status\_code = status.HTTP\_201\_CREATED

            return {"success": "Тип собственности успешно добавлен."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/positions/update/{position\_id}', status\_code=200)

def update\_position(position\_id: int, data: Positions, response: Response):

    try:

        with get\_cursor() as cursor:

            print(position\_id)

            cursor.execute(

                f"UPDATE positions SET position = '{data.position}' WHERE id = {position\_id}")

            response.status\_code = status.HTTP\_200\_OK

            return {"success": "Должность успешно изменена."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

# Care\_products

@app.get('/care\_products/getByValue', status\_code=200)

def get\_care\_product\_by\_value(care\_product: str, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                            SELECT \* FROM care\_products

                            WHERE care\_product ILIKE '%{care\_product}%'

                            ORDER BY id

                            LIMIT 50

                            """)

            care\_product = {

                "id": "",

                "care\_product": "",

                "unit": "",

            }

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                care\_product["id"] = row[0]

                care\_product["care\_product"] = row[1]

                care\_product["unit"] = row[2]

                result.append(dict(care\_product))

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/care\_products/add', status\_code=200)

def add\_care\_product(data: CareProduct, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(

                f"INSERT INTO care\_products VALUES (DEFAULT, '{data.care\_product}', '{data.unit}')")

            response.status\_code = status.HTTP\_201\_CREATED

            return {"success": f"Средство ухода c названием '{data.care\_product}' добавлено."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/care\_products/update/{care\_product\_id}', status\_code=200)

def update\_care\_product(care\_product\_id: int, data: CareProduct, response: Response):

    try:

        with get\_cursor() as cursor:

            print(care\_product\_id)

            cursor.execute(

                f"UPDATE care\_products SET care\_product = '{data.care\_product}', unit ='{data.unit}' WHERE id = {care\_product\_id}")

            response.status\_code = status.HTTP\_200\_OK

            return {"success": "Средство ухода успешно изменено."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

# Families

@app.get('/families/getByValue', status\_code=200)

def get\_family\_by\_value(family: str, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                            SELECT \* FROM families

                            WHERE family ILIKE '%{family}%'

                            ORDER BY id

                            LIMIT 50

                            """)

            family = {

                "id": "",

                "family": "",

            }

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                family["id"] = row[0]

                family["family"] = row[1]

                result.append(dict(family))

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/families/add', status\_code=200)

def add\_family(data: Families, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(

                f"INSERT INTO families VALUES (DEFAULT, '{data.family}')")

            response.status\_code = status.HTTP\_201\_CREATED

            return {"success": f"Семейcтво c названием '{data}' добавлено."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/families/update/{family\_id}', status\_code=200)

def update\_family(family\_id: int, data: Families, response: Response):

    cursor = get\_cursor()

    try:

        with get\_cursor() as cursor:

            cursor.execute(

                f"UPDATE families SET family = '{data.family}' WHERE id = {family\_id}")

            response.status\_code = status.HTTP\_200\_OK

            return {"success": "Семейcтво успешно изменено."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

# Generas

@app.get('/table/generas/getAll', status\_code=200)

def get\_generas(page: int, limit: int, response: Response):

    offset = (page - 1) \* limit

    try:

        with get\_cursor() as cursor:

            cursor.execute(f'''

                            SELECT g.id, g.genera, f.family

                            FROM generas g

                            JOIN families f ON g.family\_id = f.id

                            ORDER BY id

                            LIMIT {limit} OFFSET {offset}

                            ''')

            genera = {

                "id": "",

                "genera": "",

                "family": "",

            }

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                genera["id"] = row[0]

                genera["genera"] = row[2]

                genera["family"] = row[1]

                result.append(dict(genera))

            cursor.execute("SELECT COUNT(\*) FROM generas")

            total\_rows = cursor.fetchone()[0]

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result, "total": total\_rows}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.get('/generas/getByValue', status\_code=200)

def get\_genera\_by\_value(genera: str, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                            SELECT g.id, g.genera , f.family

                            FROM generas g

                            JOIN families f ON g.family\_id = f.id

                            WHERE genera ILIKE '%{genera}%'

                            ORDER BY id

                            LIMIT 50

                            """)

            genera = {

                "id": "",

                "genera": "",

                "family": "",

            }

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                genera["id"] = row[0]

                genera["genera"] = row[1]

                genera["family"] = row[2]

                result.append(dict(genera))

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.get('/generas/via\_family/{family\_id}', status\_code=200)

def get\_generas\_by\_family(family\_id: int, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(

                f"SELECT id, genera FROM generas WHERE family\_id = {family\_id} ORDER BY id")

            rows\_list = cursor.fetchall()

            genera = {

                "id": "",

                "genera": "",

            }

            result = []

            for row in rows\_list:

                genera["id"] = row[0]

                genera["genera"] = row[1]

                result.append(dict(genera))

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/generas/add', status\_code=200)

def add\_generas(data: Genera, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(

                f"INSERT INTO generas VALUES (default, {data.family\_id}, '{data.genera}')")

        response.status\_code = status.HTTP\_201\_CREATED

        return {"success": f"Вид c названием '{data.genera}' добавлен."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/generas/update/{genera\_id}', status\_code=200)

def update\_genera(genera\_id: int, data: Genera, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(

                f"UPDATE generas SET genera = '{data.genera}', family\_id = {data.family\_id} WHERE id = {genera\_id}")

            response.status\_code = status.HTTP\_200\_OK

            return {"success": "Средство ухода успешно изменено."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

# Plants

@app.get('/table/plants/getAll', status\_code=200)

def get\_plants(page: int, limit: int, response: Response):

    offset = (page - 1) \* limit

    try:

        with get\_cursor() as cursor:

            cursor.execute(f'''

                            SELECT p.id, p.photo, p.price, p.planting, p.death, c.full\_name, p.plot, p.conditions

                            FROM plants p

                            JOIN catalogue c ON p.catalogue\_id = c.id

                            ORDER BY p.id

                            LIMIT {limit} OFFSET {offset}

                            ''')

            plant = {

                "id": "",

                "photo": "",

                "price": "",

                "planting": "",

                "death": "",

                "catalogue": "",

                "plot": "",

                "conditions": "",

            }

            result = []

            rows\_list = cursor.fetchall()

            temp = bytearray()

            # print(psycopg2.Binary(rows\_list[0][1]))

            for row in rows\_list:

                plant["id"] = row[0]

                plant["photo"] = base64.b64encode(row[1])

                plant["price"] = row[2]

                plant["planting"] = row[3]

                plant["death"] = row[4]

                plant["catalogue"] = row[5]

                plant["plot"] = row[6]

                plant["conditions"] = row[7]

                result.append(dict(plant))

            cursor.execute("SELECT COUNT(\*) FROM plants")

            total\_rows = cursor.fetchone()[0]

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result, "total": total\_rows}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.get('/plants/getByValue', status\_code=200)

def get\_plant\_by\_value(plant: str, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                            SELECT p.id, p.photo, p.price, p.planting, p.death, c.full\_name, p.plot, p.conditions

                            FROM plants p

                            JOIN catalogue c ON p.catalogue\_id = c.id

                            WHERE full\_name ILIKE '%{plant}%'

                            ORDER BY id

                            LIMIT 50

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "id": row[0],

                    "photo": base64.b64encode(row[1]),

                    "price": row[2],

                    "planting": row[3],

                    "death": row[4],

                    "catalogue": row[5],

                    "plot": row[6],

                    "conditions": row[7],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        print(e)

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.get('/plants/get/{plant\_id}', status\_code=200)

def get\_plant(plant\_id: int, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f'''

                            SELECT p.id, p.price, p.planting, p.death, p.plot, p.conditions, c.full\_name

                            FROM plants p

                            JOIN catalogue c ON p.catalogue\_id = c.id

                            WHERE p.id = {plant\_id}

                            ORDER BY p.id

                            ''')

            plant = cursor.fetchall()

            if plant == []:

                response.status\_code = status.HTTP\_404\_NOT\_FOUND

                return {"error": "Вид не найден."}

            response.status\_code = status.HTTP\_200\_OK

            return plant

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/plants/add', status\_code=200)

def add\_plants(data: Plants, response: Response):

    print(data)

    if data.death == None:

        death = "NULL"

    else:

        death = f"'{data.death}'"

    try:

        with get\_cursor() as cursor:

            cursor.execute(

                f"INSERT INTO plants VALUES (default, '{data.photo}', {data.price}, '{data.planting}', {death}, {data.catalogue\_id}, {data.plot}, '{data.conditions}')")

            response.status\_code = status.HTTP\_201\_CREATED

            return {"success": "Растение добавлено."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/plants/update/{plant\_id}', status\_code=200)

def update\_plant(plant\_id: int, data: Plants, response: Response):

    print(data)

    if data.death == None:

        death = "NULL"

    else:

        death = f"'{data.death}'"

    try:

        with get\_cursor() as cursor:

            cursor.execute(f'''

                            UPDATE plants SET photo = '{data.photo}', price ={data.price}, planting ='{data.planting}', death ={death}, catalogue\_id ={data.catalogue\_id}, plot ={data.plot}, conditions ='{data.conditions}'

                            WHERE id = {plant\_id}

                            ''')

            response.status\_code = status.HTTP\_200\_OK

            return {"success": "Растение успешно изменено."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

# Catalogue

@app.get('/table/catalogue/getAll', status\_code=200)

def get\_catalogue(page: int, limit: int, response: Response):

    offset = (page - 1) \* limit

    try:

        with get\_cursor() as cursor:

            cursor.execute(f'''

                            SELECT c.id, c.full\_name, c.life\_span, g.genera, c.annual

                            FROM catalogue c

                            JOIN generas g ON c.genera\_id = g.id

                            ORDER BY c.id

                            LIMIT {limit} OFFSET {offset}

                            ''')

            catalogue = {

                "id": "",

                "full\_name": "",

                "life\_span": "",

                "genera": "",

                "annual": "",

            }

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                catalogue["id"] = row[0]

                catalogue["full\_name"] = row[1]

                catalogue["life\_span"] = row[2]

                catalogue["genera"] = row[3]

                catalogue["annual"] = row[4]

                result.append(dict(catalogue))

            cursor.execute("SELECT COUNT(\*) FROM catalogue")

            total\_rows = cursor.fetchone()[0]

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result, "total": total\_rows}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.get('/catalogue/getByValue', status\_code=200)

def get\_catalogue\_by\_value(catalogue: str, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                            SELECT c.id, c.full\_name, c.life\_span, g.genera, c.annual

                            FROM catalogue c

                            JOIN generas g ON c.genera\_id = g.id

                            WHERE full\_name ILIKE '%{catalogue}%'

                            ORDER BY id

                            LIMIT 50

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "id": row[0],

                    "full\_name": row[1],

                    "life\_span": row[2],

                    "genera": row[3],

                    "annual": row[4],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        print(e)

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.get('/catalogue/via\_genera/{genera\_id}', status\_code=200)

def get\_catalogue\_by\_genera(genera\_id: int, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(

                f"SELECT id, full\_name FROM catalogue WHERE genera\_id = {genera\_id} ORDER BY id")

            rows\_list = cursor.fetchall()

            catalogue = {

                "id": "",

                "full\_name": "",

            }

            result = []

            for row in rows\_list:

                catalogue["id"] = row[0]

                catalogue["full\_name"] = row[1]

                result.append(dict(catalogue))

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/catalogue/add', status\_code=200)

def add\_catalogue(data: Catalogue, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(

                f"INSERT INTO catalogue VALUES (default, '{data.full\_name}', {data.life\_span}, {data.genera\_id}, {data.annual})")

            response.status\_code = status.HTTP\_201\_CREATED

            return {"success": "Каталог добавлен."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/catalogue/update/{catalogue\_id}', status\_code=200)

def update\_catalogue(catalogue\_id: int, data: Catalogue, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f'''

                           UPDATE catalogue SET full\_name = '{data.full\_name}', life\_span = {data.life\_span}, genera\_id = {data.genera\_id}, annual = {data.annual}

                           WHERE id = {catalogue\_id}

                           ''')

            response.status\_code = status.HTTP\_200\_OK

            return {"success": "Каталог успешно изменен."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

# Employees

@app.get('/table/employees/getAll', status\_code=200)

def get\_employees(page: int, limit: int, response: Response):

    offset = (page - 1) \* limit

    try:

        with get\_cursor() as cursor:

            cursor.execute(f'''

                            SELECT e.id, e.surname, e.name, e.patronymic, e.birth, p.position, e.experience, e.salary

                            FROM employees e

                            JOIN positions p ON e.position\_id = p.id

                            ORDER BY e.id

                            LIMIT {limit} OFFSET {offset}

                            ''')

            employee = {

                "id": "",

                "surname": "",

                "name": "",

                "patronymic": "",

                "birth": "",

                "position": "",

                "experience": "",

                "salary": "",

            }

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                employee["id"] = row[0]

                employee["surname"] = row[1]

                employee["name"] = row[2]

                employee["patronymic"] = row[3]

                employee["birth"] = row[4]

                employee["position"] = row[5]

                employee["experience"] = row[6]

                employee["salary"] = row[7]

                result.append(dict(employee))

            cursor.execute("SELECT COUNT(\*) FROM employees")

            total\_rows = cursor.fetchone()[0]

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result, "total": total\_rows}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.get('/employees/getByValue', status\_code=200)

def get\_employee\_by\_value(employee: str, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                            SELECT e.id, e.surname, e.name, e.patronymic, e.birth, p.position, e.experience, e.salary

                            FROM employees e

                            JOIN positions p ON e.position\_id = p.id

                            WHERE surname ILIKE '%{employee}%'

                            ORDER BY id

                            LIMIT 50

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "id": row[0],

                    "surname": row[1],

                    "name": row[2],

                    "patronymic": row[3],

                    "birth": row[4],

                    "position": row[5],

                    "experience": row[6],

                    "salary": row[7],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        print(e)

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/employees/add', status\_code=200)

def add\_employee(data: Employees, response: Response):

    print(data)

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                           INSERT INTO employees VALUES (default,

                           '{data.surname}',

                           '{data.name}',

                           '{data.patronymic}',

                           '{data.birth}',

                           {data.position\_id},

                           {data.experience},

                           {data.salary})

                           """)

            response.status\_code = status.HTTP\_201\_CREATED

            return {"success": "Сотрудник добавлен."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/employees/update/{employee\_id}', status\_code=200)

def update\_employee(employee\_id: int, data: Employees, response: Response):

    print(data)

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                            UPDATE employees SET

                                surname = '{data.surname}',

                                name = '{data.name}',

                                patronymic = '{data.patronymic}',

                                birth = '{data.birth}',

                                position\_id = {data.position\_id},

                                experience = {data.experience},

                                salary = {data.salary}

                            WHERE id = {employee\_id}

                            """)

            response.status\_code = status.HTTP\_200\_OK

            return {"success": "Сотрудник успешно изменен."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

# Service

@app.get('/table/service/getAll', status\_code=200)

def get\_service(page: int, limit: int, response: Response):

    offset = (page - 1) \* limit

    try:

        with get\_cursor() as cursor:

            cursor.execute(f'''

                            SELECT s.id, ct.full\_name,  c.care\_product, s.amount, s.fertilized, CONCAT(e.surname, ' ', LEFT(e.name, 1), '. ', LEFT(e.patronymic, 1), '.') AS employee

                            FROM service s

                            JOIN plants p ON s.plant\_id = p.id

                            JOIN catalogue ct ON p.catalogue\_id = ct.id

                            JOIN care\_products c ON s.care\_product\_id = c.id

                            JOIN employees e ON s.employee\_id = e.id

                            ORDER BY s.id

                            LIMIT {limit} OFFSET {offset}

                            ''')

            service = {

                "id": "",

                "plant": "",

                "care\_product": "",

                "amount": "",

                "fertilized": "",

                "employee": ""

            }

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                service["id"] = row[0]

                service["plant"] = row[1]

                service["care\_product"] = row[2]

                service["amount"] = row[3]

                service["fertilized"] = row[4]

                service["employee"] = row[5]

                result.append(dict(service))

            cursor.execute("SELECT COUNT(\*) FROM service")

            total\_rows = cursor.fetchone()[0]

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result, "total": total\_rows}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.get('/service/getByValue', status\_code=200)

def get\_service\_by\_value(service: str, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                            SELECT s.id, ct.full\_name,  c.care\_product, s.amount, s.fertilized, CONCAT(e.surname, ' ', LEFT(e.name, 1), '. ', LEFT(e.patronymic, 1), '.') AS employee

                            FROM service s

                            JOIN plants p ON s.plant\_id = p.id

                            JOIN catalogue ct ON p.catalogue\_id = ct.id

                            JOIN care\_products c ON s.care\_product\_id = c.id

                            JOIN employees e ON s.employee\_id = e.id

                            WHERE full\_name ILIKE '%{service}%'

                            ORDER BY s.id

                            LIMIT 50

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "id": row[0],

                    "plant": row[1],

                    "care\_product": row[2],

                    "amount": row[3],

                    "fertilized": row[4],

                    "employee": row[5]

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        print(e)

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/service/add', status\_code=200)

def add\_service(data: Service, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f'''

                           INSERT INTO service VALUES (default, {data.plant\_id}, {data.care\_product\_id}, {data.amount}, '{data.fertilized}', {data.employee\_id})

                           ''')

            response.status\_code = status.HTTP\_201\_CREATED

            return {"success": "Обслуживание добавлено."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/service/update/{service\_id}', status\_code=200)

def update\_service(service\_id: int, data: Service, response: Response):

    print(data)

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                            UPDATE service SET

                            plant\_id = {data.plant\_id},

                            care\_product\_id = {data.care\_product\_id},

                            amount = {data.amount},

                            fertilized = '{data.fertilized}',

                            employee\_id = {data.employee\_id}

                            WHERE id = {service\_id}

                            """)

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        print()

        print(e)

        return {"error": f"Возникла необработанная ошибка: {e}"}

# Gardens

@app.get('/table/gardens/getAll', status\_code=200)

def get\_gardens(page: int, limit: int, response: Response):

    offset = (page - 1) \* limit

    try:

        with get\_cursor() as cursor:

            cursor.execute(f'''

                            SELECT g.id, g.garden, g.opening, g.phone, g.financing, c.city, p.property, e.surname, e.name

                            FROM gardens g

                            JOIN cities c ON g.city\_id = c.id

                            JOIN properties p ON g.property\_id = p.id

                            JOIN employees e ON g.employee\_id = e.id

                            ORDER BY id

                            LIMIT {limit} OFFSET {offset}

                            ''')

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "id": row[0],

                    "garden": row[1],

                    "opening": row[2],

                    "phone": row[3],

                    "financing": row[4],

                    "city": row[5],

                    "property": row[6],

                    "employee": row[7] + " " + row[8]

                })

            cursor.execute("SELECT COUNT(\*) FROM gardens")

            total\_rows = cursor.fetchone()[0]

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result, "total": total\_rows}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        print(e)

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.get('/gardens/getByValue', status\_code=200)

def get\_garden\_by\_value(garden: str, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                            SELECT g.id, g.garden, g.opening, g.phone, g.financing, c.city, p.property, e.surname, e.name

                            FROM gardens g

                            JOIN cities c ON g.city\_id = c.id

                            JOIN properties p ON g.property\_id = p.id

                            JOIN employees e ON g.employee\_id = e.id

                            WHERE garden ILIKE '%{garden}%'

                            ORDER BY id

                            LIMIT 50

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "id": row[0],

                    "garden": row[1],

                    "opening": row[2],

                    "phone": row[3],

                    "financing": row[4],

                    "city": row[5],

                    "property": row[6],

                    "employee": row[7] + " " + row[8]

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        print(e)

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/gardens/add', status\_code=200)

def add\_garden(data: Gardens, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                           INSERT INTO gardens VALUES (default,

                           '{data.garden}',

                           {data.opening},

                           {data.phone},

                           {data.financing},

                           {data.city\_id},

                           {data.position\_id},

                           {data.employee\_id})

                           """)

            response.status\_code = status.HTTP\_201\_CREATED

            return {"success": "Сад добавлен."}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@app.post('/gardens/update/{garden\_id}', status\_code=200)

def update\_garden(garden\_id: int, data: Gardens, response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                            UPDATE gardens SET

                            garden = '{data.garden}',

                            opening = {data.opening},

                            phone = {data.phone},

                            financing = {data.financing},

                            city\_id = {data.city\_id},

                            property\_id = {data.property\_id},

                            employee\_id = {data.employee\_id}

                            WHERE id = {garden\_id}

                            """)

            response.status\_code = status.HTTP\_200\_OK

            return {"success": "Сад успешно изменен."}

    except Exception as e:

        print(e)

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

import psycopg2

from fastapi import FastAPI, Response, status

from fastapi import APIRouter

from db\_models import \*

router = APIRouter()

def get\_cursor():

    conn = psycopg2.connect(

        dbname='Kurs',

        user='postgres',

        password='0988',

        host='localhost',

        port='5434'

    )

    cursor = conn.cursor()

    conn.autocommit = True

    return cursor

@router.get("/queries/1", status\_code=200)

def query\_1(response: Response):

    try:

        with get\_cursor() as cursor:

            # Все сады с типом собственности "Смешанная собственность"

            cursor.execute(f"""

                            SELECT g.id, g.garden, p.property

                            FROM gardens g

                            INNER JOIN properties p ON g.property\_id = p.id

                            WHERE g.property\_id = 3

                            ORDER BY g.id

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "КОД": row[0],

                    "Сад": row[1],

                    "Тип собственности": row[2],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/2", status\_code=200)

def query\_2(response: Response):

    try:

        with get\_cursor() as cursor:

            # Все сады в городе "Санкт-Петербург"

            cursor.execute(f"""

                            SELECT g.id, g.garden, c.city

                            FROM gardens g

                            INNER JOIN cities c ON g.city\_id = c.id

                            WHERE g.city\_id = 2

                            ORDER BY g.id

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "КОД": row[0],

                    "Сад": row[1],

                    "Город": row[2],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/3", status\_code=200)

def query\_3(response: Response):

    try:

        with get\_cursor() as cursor:

            # Список ухода за последниюю неделю

            cursor.execute(f"""

                            SELECT s.id, ct.full\_name, s.fertilized, CONCAT(e.surname, ' ', LEFT(e.name, 1), '. ', LEFT(e.patronymic, 1), '.') AS employee

                            FROM service s

                            JOIN plants p ON s.plant\_id = p.id

                            JOIN catalogue ct ON p.catalogue\_id = ct.id

                            JOIN employees e ON s.employee\_id = e.id

                            WHERE s.fertilized >= CURRENT\_date - 7

                            ORDER BY s.id

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "КОД": row[0],

                    "Полное имя": row[1],

                    "Дата удобрения": row[2],

                    "ФИО сотрудника": row[3],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/4", status\_code=200)

def query\_4(response: Response):

    try:

        with get\_cursor() as cursor:

            # Список ухода за последний сезон

            cursor.execute(f"""

                            SELECT s.id, ct.full\_name, s.fertilized, CONCAT(e.surname, ' ', LEFT(e.name, 1), '. ', LEFT(e.patronymic, 1), '.') AS employee

                            FROM service s

                            JOIN plants p ON s.plant\_id = p.id

                            JOIN catalogue ct ON p.catalogue\_id = ct.id

                            JOIN employees e ON s.employee\_id = e.id

                            WHERE s.fertilized >= CURRENT\_date - 90

                            ORDER BY s.id

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "КОД": row[0],

                    "Полное имя": row[1],

                    "Дата удобрения": row[2],

                    "ФИО сотрудника": row[3],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/5", status\_code=200)

def query\_5(response: Response):

    try:

        with get\_cursor() as cursor:

            # Симметричное внутреннее соединение без условия

            cursor.execute(f"""

                            SELECT e.id, CONCAT(e.surname, ' ', LEFT(e.name, 1), '. ', LEFT(e.patronymic, 1), '.') AS employee, p.position

                            FROM employees e

                            JOIN positions p ON e.position\_id = p.id

                            ORDER BY e.id

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "КОД": row[0],

                    "Полное имя": row[1],

                    "Должность": row[2],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/6", status\_code=200)

def query\_6(response: Response):

    try:

        with get\_cursor() as cursor:

            # Симметричное внутреннее соединение без условия

            cursor.execute(f"""

                            SELECT g.id, g.garden, p.property

                            FROM gardens g

                            INNER JOIN properties p ON g.property\_id = p.id

                            ORDER BY g.id

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "КОД": row[0],

                    "Название": row[1],

                    "Тип собственности": row[2],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/7", status\_code=200)

def query\_7(response: Response):

    try:

        with get\_cursor() as cursor:

            # Симметричное внутреннее соединение без условия

            cursor.execute(f"""

                            SELECT g.id, g.garden, c.city

                            FROM gardens g

                            INNER JOIN cities c ON g.city\_id = c.id

                            ORDER BY g.id

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "КОД": row[0],

                    "Название": row[1],

                    "Город": row[2],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/8", status\_code=200)

def query\_8(response: Response):

    try:

        with get\_cursor() as cursor:

            # Сотрудники, участвующие в уходе

            cursor.execute(f"""

                            SELECT e.id, CONCAT(e.surname, ' ', LEFT(e.name, 1), '. ', LEFT(e.patronymic, 1), '.') AS employee, p.position, s.fertilized

                            FROM employees e

                            JOIN positions p ON e.position\_id = p.id

                            LEFT JOIN service s ON s.employee\_id = e.id

                            ORDER BY e.id

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "КОД": row[0],

                    "ФИО": row[1],

                    "Должность": row[2],

                    "Дата удобрения": row[3],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/9", status\_code=200)

def query\_9(response: Response):

    try:

        with get\_cursor() as cursor:

            # Сотрудники, ухаживающие за еще не умершими растениями

            cursor.execute(f"""

                            SELECT e.id, CONCAT(e.surname, ' ', LEFT(e.name, 1), '. ', LEFT(e.patronymic, 1), '.') AS employee, c.full\_name, p.death

                            FROM employees e

                            RIGHT JOIN service s ON s.employee\_id = e.id

                            RIGHT JOIN plants p ON s.plant\_id = p.id

                            JOIN catalogue c ON p.catalogue\_id = c.id

                            WHERE p.death is NULL

                            ORDER BY e.id

                            LIMIT 50

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "КОД": row[0],

                    "ФИО": row[1],

                    "Название растения": row[2],

                    "Дата смерти": "null" if row[3] is None else row[3],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/10", status\_code=200)

def query\_10(response: Response):

    try:

        with get\_cursor() as cursor:

            # Вывод информации о количестве использованного удобрения за сезон

            cursor.execute(f"""

                            SELECT DISTINCT c.care\_product,

                            (SELECT COUNT(\*)

                            FROM service s

                            WHERE s.care\_product\_id = c.id AND s.fertilized >= CURRENT\_date - 90) as "Потрачено"

                            FROM care\_products c

                            LEFT JOIN service s ON s.care\_product\_id = c.id

                            ORDER BY "Потрачено" DESC;

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "Средство ухода": row[0],

                    "Израсходовано": row[1],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/11", status\_code=200)

def query\_11(response: Response):

    try:

        with get\_cursor() as cursor:

            # Вывод количества видов в семействах

            cursor.execute(f"""

                            SELECT f.family, COUNT(\*) as "Виды"

                            FROM generas g

                            INNER JOIN families f ON g.family\_id = f.id

                            GROUP BY f.family

                            ORDER BY "Виды" DESC

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "Семейство": row[0],

                    "Кол-во видов": row[1],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/12", status\_code=200)

def query\_12(response: Response):

    try:

        with get\_cursor() as cursor:

            # Вывод информации о количестве садов с финансированием и без

            cursor.execute(f"""

                            SELECT count(\*), count(CASE WHEN financing = true THEN id END),

                            count(CASE WHEN financing = false THEN id END)

                            FROM gardens

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "Общее кол-во садов": row[0],

                    "С финансированием": row[1],

                    "Без финансирования": row[2],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/13", status\_code=200)

def query\_13(response: Response):

    try:

        with get\_cursor() as cursor:

            # Количество опр. типа собственности за период после 2010г.

            cursor.execute(f"""

                            SELECT p.property, COUNT(g.property\_id)

                            FROM gardens g

                            INNER JOIN properties p ON g.property\_id = p.id

                            WHERE opening > 2010

                            GROUP BY p.property

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "Тип собственности": row[0],

                    "Кол-во садов": row[1],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/14", status\_code=200)

def query\_14(response: Response):

    try:

        with get\_cursor() as cursor:

            # Количество растений, обслуживаемых Владимирами

            cursor.execute(f"""

                            SELECT e.id, e.surname, e.name, count(s.\*)

                            FROM service s

                            JOIN employees e ON s.employee\_id = e.id

                            GROUP BY e.id, e.surname, e.name

                            HAVING e.name ILIKE 'Владимир%'

                            ORDER BY e.id ASC

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "КОД": row[0],

                    "Фамилия": row[1],

                    "Имя": row[2],

                    "Кол-во": row[3],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/15", status\_code=200)

def query\_15(response: Response):

    try:

        with get\_cursor() as cursor:

            # Получить количество садов в каждом городе и указать максимальное открытие среди них

            cursor.execute(f"""

                            SELECT

                                c.city,

                                COUNT(g.id) AS garden\_count,

                                MAX(g.opening) AS max\_opening

                            FROM gardens g

                            JOIN cities c ON g.city\_id = c.id

                            GROUP BY c.city;

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "Город": row[0],

                    "Кол-во садов": row[1],

                    "Самое позднее открытие": row[2],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/16", status\_code=200)

def query\_16(response: Response):

    try:

        with get\_cursor() as cursor:

            # Количество растений определенного вида

            cursor.execute(f"""

                            SELECT g.genera, COUNT(\*) as "Кол-во растений"

                            FROM plants p

                            JOIN catalogue c ON p.catalogue\_id = c.id

                            JOIN generas g ON c.genera\_id = g.id

                            GROUP BY p.death, g.genera

                            HAVING p.death is NULL

                            ORDER BY "Кол-во растений" DESC

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "Вид": row[0],

                    "Кол-во": row[1],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/17", status\_code=200)

def query\_17(response: Response):

    try:

        with get\_cursor() as cursor:

            # Количество растений определенного вида, которые вымерли

            cursor.execute(f"""

                            SELECT g.genera, COUNT(\*) as "Кол-во растений"

                            FROM plants p

                            JOIN catalogue c ON p.catalogue\_id = c.id

                            JOIN generas g ON c.genera\_id = g.id

                            WHERE p.death IS NOT NULL

                            GROUP BY g.genera

                            ORDER BY "Кол-во растений" DESC

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "Вид": row[0],

                    "Кол-во": row[1],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/18", status\_code=200)

def query\_18(response: Response):

    try:

        with get\_cursor() as cursor:

            # Cредняя стоимость саженцев не превышает 888

            cursor.execute(f"""

                            SELECT c.full\_name, round(avg(p.price), 3) as "AVG"

                            FROM plants p

                            INNER JOIN catalogue c ON p.catalogue\_id = c.id

                            GROUP BY c.full\_name

                            HAVING avg(p.price) < 888

                            ORDER BY "AVG" DESC

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "Название": row[0],

                    "Средняя цена": row[1],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/19", status\_code=200)

def query\_19(response: Response):

    try:

        with get\_cursor() as cursor:

            # Количество растений за удобренных за 90 дней, превышающих 2

            cursor.execute(f"""

                            SELECT CONCAT(e.surname, ' ', LEFT(e.name, 1), '. ', LEFT(e.patronymic, 1), '.') AS employee, COUNT(s.\*)

                            FROM service s

                            RIGHT JOIN employees e ON s.employee\_id = e.id

                            WHERE s.fertilized >= CURRENT\_date - 90

                            GROUP BY employee

                            HAVING count(s.id) > 2

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "ФИО": row[0],

                    "Кол-во": row[1],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/20", status\_code=200)

def query\_20(response: Response):

    try:

        with get\_cursor() as cursor:

            # Кол-во политых растениях каждым сотрудником

            cursor.execute(f"""

                            SELECT DISTINCT

                            CONCAT(e.surname, ' ', LEFT(e.name, 1), '. ', LEFT(e.patronymic, 1), '.') AS employee,

                            e.birth,

                            (SELECT COUNT(\*) FROM service s WHERE s.employee\_id = e.id) AS "1" FROM employees e

                            ORDER BY "1" DESC;

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "ФИО": row[0],

                    "Дата рождения": row[1],

                    "Кол-во": row[2],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/21", status\_code=200)

def query\_21(response: Response):

    try:

        with get\_cursor() as cursor:

            # Инф. о Работниках с опытом работы более 2 лет и их обслуживании.

            cursor.execute(f"""

                            SELECT id, CONCAT(employees.surname, ' ', LEFT(employees.name, 1), '. ', LEFT(employees.patronymic, 1), '.') AS employee,

                            experience, NULL AS service\_info

                            FROM employees

                            WHERE experience > 2

                            UNION

                            SELECT employees.id, CONCAT(employees.surname, ' ', LEFT(employees.name, 1), '. ', LEFT(employees.patronymic, 1), '.') AS employee,

                            employees.experience,

                            service.amount || ' разных растений обслуживается ' AS service\_info

                            FROM employees

                            JOIN service ON employees.id = service.employee\_id

                            JOIN plants ON service.plant\_id = plants.id

                            WHERE employees.experience > 2;

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "КОД": row[0],

                    "ФИО": row[1],

                    "Опыт работы": row[2],

                    "Обслуживание": "null" if row[3] is None else row[3],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/22", status\_code=200)

def query\_22(response: Response):

    try:

        with get\_cursor() as cursor:

            # Дата удобрения растения с сроком жизни больше 5 лет

            cursor.execute(f"""

                            SELECT

                            s.fertilized,

                            cat.full\_name,

                            cat.life\_span

                            FROM service s

                            JOIN plants p ON s.plant\_id = p.id

                            JOIN catalogue cat ON p.catalogue\_id = cat.id

                            WHERE cat.life\_span > 5

                            AND p.id IN (SELECT plant\_id FROM service);

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "Дата": row[0],

                    "Полное название": row[1],

                    "Срок жизни": row[2],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/23", status\_code=200)

def query\_23(response: Response):

    try:

        with get\_cursor() as cursor:

            # Растения без обслуживания и обслуживающих сотрудников

            cursor.execute(f"""

                            SELECT

                                p.id,

                                c.full\_name,

                                p.planting,

                                p.death

                            FROM plants p

                            JOIN catalogue c ON p.catalogue\_id = c.id

                            LEFT JOIN service s ON p.id = s.plant\_id

                            LEFT JOIN employees e ON s.employee\_id = e.id

                            WHERE p.id NOT IN (SELECT plant\_id FROM service WHERE plant\_id IS NOT NULL)

                            ORDER BY p.id

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "КОД": row[0],

                    "Полное название": row[1],

                    "Дата посадки": row[2],

                    "Дата гибели": row[3],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/24", status\_code=200)

def query\_24(response: Response):

    try:

        with get\_cursor() as cursor:

            # Получить список сотрудников, их должности и указать "Молодой" или "Опытный" в зависимости от опыта работы

            cursor.execute(f"""

                            SELECT

                                e.id AS employee\_id,

                                CONCAT(e.surname, ' ', LEFT(e.name, 1), '. ', LEFT(e.patronymic, 1), '.') AS employee,

                                p.position,

                                e.experience,

                                CASE

                                    WHEN e.experience <= 5 THEN 'Молодой'

                                    ELSE 'Опытный'

                                END AS experience\_category

                            FROM employees e

                            JOIN positions p ON e.position\_id = p.id;

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "КОД": row[0],

                    "ФИО": row[1],

                    "Должность": row[2],

                    "Опыт": row[3],

                    "Оценка опыта": row[4],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/25", status\_code=200)

def query\_25(response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                            SELECT g.garden,

                            COUNT(p.\*) AS percentage

                            FROM plants p

                            RIGHT JOIN service s ON p.id = s.plant\_id

                            INNER JOIN employees e ON s.employee\_id = e.id

                            INNER JOIN gardens g ON e.id = g.employee\_id

                            WHERE p.planting < '2020-01-01' GROUP BY g.garden;

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "garden": row[0],

                    "percentage": row[1],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/26", status\_code=200)

def query\_26(response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                            SELECT g.garden as "Сад",

                            round (avg(extract('year' FROM age (current\_date, e.birth)))) as "Cредний возраст сотрудников",

                            round (avg(e.experience), 1) as "Средний стаx сотрудников"

                            FROM gardens g

                            JOIN employees e ON g.employee\_id = e.id

                            GROUP BY g.garden;

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "Сад": row[0],

                    "Cредний возраст сотрудников": row[1],

                    "Средний стаx сотрудников": row[2],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

@router.get("/queries/27", status\_code=200)

def query\_27(response: Response):

    try:

        with get\_cursor() as cursor:

            cursor.execute(f"""

                            SELECT g.garden,

                            COUNT(p.\*) AS "Количество растений", SUM((SELECT s.amount)) AS "Расходы по уходу за растениями"

                            FROM service s

                            RIGHT JOIN plants p ON s.plant\_id = p.id

                            INNER JOIN employees e ON s.employee\_id = e.id

                            INNER JOIN gardens g ON e.id = g.employee\_id

                            GROUP BY g.garden;

                            """)

            result = []

            rows\_list = cursor.fetchall()

            for row in rows\_list:

                result.append({

                    "garden": row[0],

                    "Количество растений": row[1],

                    "Расходы по уходу за растениями": row[2],

                })

            response.status\_code = status.HTTP\_200\_OK

            return {"rows": result}

    except Exception as e:

        response.status\_code = status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

        return {"error": f"Возникла необработанная ошибка: {e}"}

from typing import Optional

from pydantic import BaseModel, validator

# Справочники

class Cities(BaseModel):

    city: str

class Properties(BaseModel):

    property: str

class Positions(BaseModel):

    position: str

class Families(BaseModel):

    family: str

class CareProduct(BaseModel):

    care\_product: str

    unit: str

# Таблицы

class Genera(BaseModel):

    genera: str

    family\_id: int

class Plants(BaseModel):

    photo: str # bytea

    price: int

    planting: str       # date

    death: Optional[str]         # date

    catalogue\_id: int

    plot: int

    conditions: str

class Catalogue(BaseModel):

    full\_name: str

    life\_span: int

    genera\_id: int

    annual: bool

class Employees(BaseModel):

    surname: str

    name: str

    patronymic: str

    birth: str      # date

    position\_id: int

    experience: int

    salary: int

class Gardens(BaseModel):

    garden: str

    opening: int

    phone: int

    financing: bool

    city\_id: int

    property\_id: int

    employee\_id: int

class Service(BaseModel):

    plant\_id: int

    care\_product\_id: int

    amount: int

    fertilized: str     # date

    employee\_id: int

ПРИЛОЖЕНИЕ Б.  
ЛИСТИНГ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

using DataBaseKursRabota.Models;

using DataBaseKursRabota.Presenters;

using DataBaseKursRabota.Repositories;

using DataBaseKursRabota.Views;

namespace DataBaseKursRabota

{

internal static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

// To customize application configuration such as set high DPI settings or default font,

// see https://aka.ms/applicationconfiguration.

ApplicationConfiguration.Initialize();

//IMainView view = new MainView();

//new MainPresenter(view);

//Application.Run((Form)view);

Application.Run(new LoginForm());

}

}

}

namespace DataBaseKursRabota.Models

{

public interface IQueryRepository

{

IEnumerable<Dictionary<string, object>> Query(int number);

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models

{

public class CareProductModel

{

private int id;

private string careProduct;

private string unit;

[JsonProperty("id")] public int Id { get => id; set => id = value; }

[JsonProperty("care\_product")] public string CareProduct { get => careProduct; set => careProduct = value; }

[JsonProperty("unit")] public string Unit { get => unit; set => unit = value; }

public override string ToString()

{

return CareProduct;

}

}

}

using Newtonsoft.Json;

namespace DataBaseKursRabota.Models

{

public class CityModel

{

private int id;

private string city;

[JsonProperty("id")] public int Id { get => id; set => id = value; }

[JsonProperty("city")] public string City { get => city; set => city = value; }

public override string ToString()

{

return City;

}

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models

{

public class FamilyModel

{

private int id;

private string family;

[JsonProperty("id")] public int Id { get => id; set => id = value; }

[JsonProperty("family")] public string Family { get => family; set => family = value; }

public override string ToString()

{

return Family;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models

{

public interface ICareProductRepository

{

void Add(CareProductModel dataModel);

void Edit(CareProductModel dataModel);

void Delete(int id);

int GetTotal();

IEnumerable<CareProductModel> GetAll(int page, int limit);

IEnumerable<CareProductModel> GetByValue(string value);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models

{

public interface ICityRepository

{

void Add(CityModel dataModel);

void Edit(CityModel dataModel);

void Delete(int id);

int GetTotal();

IEnumerable<CityModel> GetAll(int page, int limit);

IEnumerable<CityModel> GetByValue(string value);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models

{

public interface IFamilyRepository

{

void Add(FamilyModel dataModel);

void Edit(FamilyModel dataModel);

void Delete(int id);

int GetTotal();

IEnumerable<FamilyModel> GetAll(int page, int limit);

IEnumerable<FamilyModel> GetByValue(string value);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models

{

public interface IPositionRepository

{

void Add(PositionModel dataModel);

void Edit(PositionModel dataModel);

void Delete(int id);

int GetTotal();

IEnumerable<PositionModel> GetAll(int page, int limit);

IEnumerable<PositionModel> GetByValue(string value);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models

{

public interface IPropertyRepository

{

void Add(PropertyModel dataModel);

void Edit(PropertyModel dataModel);

void Delete(int id);

int GetTotal();

IEnumerable<PropertyModel> GetAll(int page, int limit);

IEnumerable<PropertyModel> GetByValue(string value);

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models

{

public class PositionModel

{

private int id;

private string position;

[JsonProperty("id")] public int Id { get => id; set => id = value; }

[JsonProperty("position")] public string Position { get => position; set => position = value; }

public override string ToString()

{

return Position;

}

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models

{

public class PropertyModel

{

private int id;

private string property;

[JsonProperty("id")] public int Id { get => id; set => id = value; }

[JsonProperty("property")] public string Property { get => property; set => property = value; }

public override string ToString()

{

return Property;

}

}

}

using Newtonsoft.Json;

namespace DataBaseKursRabota.Models.Tables

{

public class CatalogueDataModel

{

private int id;

private string fullname;

private int lifeSpan;

private string genera;

private int generaId;

private bool annual;

[JsonProperty("id")] public int Id { get => id; set => id = value; }

[JsonProperty("full\_name")] public string Fullname { get => fullname; set => fullname = value; }

[JsonProperty("life\_span")] public int LifeSpan { get => lifeSpan; set => lifeSpan = value; }

[JsonProperty("genera")] public string Genera { get => genera; set => genera = value; }

[JsonProperty("genera\_id")] public int GeneraId { get => generaId; set => generaId = value; }

[JsonProperty("annual")] public bool Annual { get => annual; set => annual = value; }

public override string ToString()

{

return Fullname;

}

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models.Tables

{

public class EmployeeDataModel

{

private int id;

private string surname;

private string name;

private string patronymic;

private string birth;

private string position;

private int positionId;

private int experience;

private int salary;

[JsonProperty("id")] public int Id { get => id; set => id = value; }

[JsonProperty("surname")] public string Surname { get => surname; set => surname = value; }

[JsonProperty("name")] public string Name { get => name; set => name = value; }

[JsonProperty("patronymic")] public string Patronymic { get => patronymic; set => patronymic = value; }

[JsonProperty("birth")] public string Birth { get => birth; set => birth = value; }

[JsonProperty("position")] public string Position { get => position; set => position = value; }

[JsonProperty("position\_id")] public int PositionId { get => positionId; set => positionId = value; }

[JsonProperty("experience")] public int Experience { get => experience; set => experience = value; }

[JsonProperty("salary")] public int Salary { get => salary; set => salary = value; }

public override string ToString()

{

return Surname + " " + Name + " " + Patronymic;

}

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models.Tables

{

public class GardenDataModel

{

private int id;

private string garden;

private int opening;

private Int64 phone;

private bool financing;

private string city;

private int cityId;

private string property;

private int propertyId;

private string employee;

private int employeeId;

[JsonProperty("id")] public int Id { get => id; set => id = value; }

[JsonProperty("garden")] public string Garden { get => garden; set => garden = value; }

[JsonProperty("opening")] public int Opening { get => opening; set => opening = value; }

[JsonProperty("phone")] public Int64 Phone { get => phone; set => phone = value; }

[JsonProperty("financing")] public bool Financing { get => financing; set => financing = value; }

[JsonProperty("city")] public string City { get => city; set => city = value; }

[JsonProperty("city\_id")] public int CityId { get => cityId; set => cityId = value; }

[JsonProperty("property")] public string Property { get => property; set => property = value; }

[JsonProperty("property\_id")] public int PropertyId { get => propertyId; set => propertyId = value; }

[JsonProperty("employee")] public string Employee { get => employee; set => employee = value; }

[JsonProperty("employee\_id")] public int EmployeeId { get => employeeId; set => employeeId = value; }

public override string ToString()

{

return Garden;

}

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models.Tables

{

public class GeneraDataModel

{

private int id;

private string genera;

private string family;

private int familyId;

[JsonProperty("id")] public int Id { get => id; set => id = value; }

[JsonProperty("genera")] public string Genera { get => genera; set => genera = value; }

[JsonProperty("family")] public string Family { get => family; set => family = value; }

[JsonProperty("family\_id")] public int FamilyId { get => familyId; set => familyId = value; }

public override string ToString()

{

return Genera;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models.Tables

{

public interface ICatalogueRepository

{

void Add(CatalogueDataModel catalogueModel);

void Edit(CatalogueDataModel catalogueModel);

void Delete(int id);

int GetTotal();

IEnumerable<CatalogueDataModel> GetAll(int page, int limit);

IEnumerable<CatalogueDataModel> GetByValue(string value);

}

}

namespace DataBaseKursRabota.Models.Tables

{

public interface IEmployeeRepository

{

void Add(EmployeeDataModel employeeModel);

Task AddEmployeeAsync(EmployeeDataModel employeeModel);

void Edit(EmployeeDataModel employeeModel);

void Delete(int id);

int GetTotal();

EmployeeDataModel Generate(IEnumerable<PositionModel> positionList);

IEnumerable<EmployeeDataModel> GetAll(int page, int limit);

IEnumerable<EmployeeDataModel> GetByValue(string value);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models.Tables

{

public interface IGardenRepository

{

void Add(GardenDataModel gardenModel);

void Edit(GardenDataModel gardenModel);

void Delete(int id);

int GetTotal();

IEnumerable<GardenDataModel> GetAll(int page, int limit);

IEnumerable<GardenDataModel> GetByValue(string value);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models.Tables

{

public interface IGeneraRepository

{

void Add(GeneraDataModel generaModel);

void Edit(GeneraDataModel generaModel);

void Delete(int id);

int GetTotal();

IEnumerable<GeneraDataModel> GetAll(int page, int limit);

IEnumerable<GeneraDataModel> GetByValue(string value);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models.Tables

{

public interface IPlantRepository

{

void Add(PlantDataModel plantModel);

void Edit(PlantDataModel plantModel);

void Delete(int id);

int GetTotal();

IEnumerable<PlantDataModel> GetAll(int page, int limit);

IEnumerable<PlantDataModel> GetByValue(string value);

}

}

namespace DataBaseKursRabota.Models.Tables

{

public interface IServiceRepository

{

void Add(ServiceDataModel serviceModel);

Task AddServiceAsync(ServiceDataModel serviceModel);

void Edit(ServiceDataModel serviceModel);

void Delete(int id);

int GetTotal();

ServiceDataModel Generate(IEnumerable<CareProductModel> careProductList, IEnumerable<EmployeeDataModel> employeeList, IEnumerable<PlantDataModel> plantList);

IEnumerable<ServiceDataModel> GetAll(int page, int limit);

IEnumerable<ServiceDataModel> GetByValue(string value);

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models.Tables

{

public class PlantDataModel

{

private int id;

private string photo;

private int price;

private string planting;

private string death;

private string catalogue;

private int catalogueId;

private int plot;

private string conditions;

[JsonProperty("id")] public int Id { get => id; set => id = value; }

public Image Photo

{

get

{

if (IsBase64Image(Image))

{

byte[] bytes = Convert.FromBase64String(Image);

Image image;

using (MemoryStream ms = new MemoryStream(bytes))

{

image = System.Drawing.Image.FromStream(ms);

}

return image;

}

else

{

// Обработка случая, когда Photo не является Base64-изображением.

return null;

}

}

set

{

byte[] bytes = null;

if (value != null)

{

ImageConverter converter = new ImageConverter();

bytes = converter.ConvertTo(value, typeof(byte[])) as byte[];

}

Image = "\\x" + BitConverter.ToString(bytes).Replace("-", "");

}

}

[JsonProperty("photo")] public string Image { get => photo; set => photo = value; }

[JsonProperty("price")] public int Price { get => price; set => price = value; }

[JsonProperty("planting")] public string Planting { get => planting; set => planting = value; }

[JsonProperty("death")] public string Death { get => death; set => death = value; }

[JsonProperty("catalogue")] public string Catalogue { get => catalogue; set => catalogue = value; }

[JsonProperty("catalogue\_id")] public int CatalogueId { get => catalogueId; set => catalogueId = value; }

[JsonProperty("plot")] public int Plot { get => plot; set => plot = value; }

[JsonProperty("conditions")] public string Conditions { get => conditions; set => conditions = value; }

public override string ToString()

{

return Price.ToString() + "/" + Catalogue + "/" + Plot.ToString();

}

public bool IsBase64Image(string base64String)

{

try

{

byte[] bytes = Convert.FromBase64String(base64String);

using (MemoryStream ms = new MemoryStream(bytes))

{

System.Drawing.Image.FromStream(ms);

return true; // Если конвертация прошла успешно, то это Base64-изображение.

}

}

catch (Exception)

{

return false; // Если возникла ошибка при конвертации, то это не Base64-изображение.

}

}

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Models.Tables

{

public class ServiceDataModel

{

private int id;

private string careProduct;

private int careProductId;

private int amount;

private string fertilized;

private string employee;

private int employeeId;

private string plant;

private int plantId;

[JsonProperty("id")] public int Id { get => id; set => id = value; }

[JsonProperty("care\_product")] public string CareProduct { get => careProduct; set => careProduct = value; }

[JsonProperty("care\_product\_id")] public int CareProductId { get => careProductId; set => careProductId = value; }

[JsonProperty("amount")] public int Amount { get => amount; set => amount = value; }

[JsonProperty("fertilized")] public string Fertilized { get => fertilized; set => fertilized = value; }

[JsonProperty("employee")] public string Employee { get => employee; set => employee = value; }

[JsonProperty("employee\_id")] public int EmployeeId { get => employeeId; set => employeeId = value; }

[JsonProperty("plant")] public string Plant { get => plant; set => plant = value; }

[JsonProperty("plant\_id")] public int PlantId { get => plantId; set => plantId = value; }

}

}

// <autogenerated />

using System;

using System.Reflection;

[assembly: global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETCoreApp,Version=v7.0", FrameworkDisplayName = ".NET 7.0")]

//------------------------------------------------------------------------------

// <auto-generated>

// Этот код создан программой.

// Исполняемая версия:4.0.30319.42000

//

// Изменения в этом файле могут привести к неправильной работе и будут потеряны в случае

// повторной генерации кода.

// </auto-generated>

//------------------------------------------------------------------------------

using System;

using System.Reflection;

[assembly: System.Reflection.AssemblyCompanyAttribute("DataBaseKursRabota")]

[assembly: System.Reflection.AssemblyConfigurationAttribute("Debug")]

[assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]

[assembly: System.Reflection.AssemblyInformationalVersionAttribute("1.0.0")]

[assembly: System.Reflection.AssemblyProductAttribute("DataBaseKursRabota")]

[assembly: System.Reflection.AssemblyTitleAttribute("DataBaseKursRabota")]

[assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]

[assembly: System.Runtime.Versioning.TargetPlatformAttribute("Windows7.0")]

[assembly: System.Runtime.Versioning.SupportedOSPlatformAttribute("Windows7.0")]

// Создано классом WriteCodeFragment MSBuild.

// <auto-generated/>

global using global::System;

global using global::System.Collections.Generic;

global using global::System.Drawing;

global using global::System.IO;

global using global::System.Linq;

global using global::System.Net.Http;

global using global::System.Threading;

global using global::System.Threading.Tasks;

global using global::System.Windows.Forms;

using DataBaseKursRabota.Models;

using DataBaseKursRabota.Models.Tables;

using DataBaseKursRabota.Presenters.Tables;

using DataBaseKursRabota.Repositories;

using DataBaseKursRabota.Repositories.Tables;

using DataBaseKursRabota.Views;

using DataBaseKursRabota.Views.References;

using DataBaseKursRabota.Views.Tables;

namespace DataBaseKursRabota.Presenters

{

public class MainPresenter

{

private IMainView mainView;

public MainPresenter(IMainView mainView)

{

this.mainView = mainView;

//Справочники

this.mainView.ShowCityView += ShowCitiesView;

this.mainView.ShowFamilyView += ShowFamilyView;

this.mainView.ShowPropertyView += ShowPropertyView;

this.mainView.ShowPositionView += ShowPositionView;

this.mainView.ShowCareProductView += ShowCareProductView;

//Таблицы

this.mainView.ShowGeneraView += ShowGeneraView;

this.mainView.ShowCatalogueView += ShowCatalogueView;

this.mainView.ShowEmployeeView += ShowEmployeeView;

this.mainView.ShowPlantView += ShowPlantView;

this.mainView.ShowGardenView += ShowGardenProductView;

this.mainView.ShowServiceView += ShowServiceView;

//Запросы

this.mainView.ShowQueriesView += ShowQueriesView;

this.mainView.Show();

}

//Запросы

private void ShowQueriesView(object? sender, EventArgs e)

{

IQueryView view = QueryView.GetInstace((MainView)mainView);

IQueryRepository repository = new QueryRepository();

new QueryPresenter(view, repository);

}

//Таблицы

private void ShowServiceView(object? sender, EventArgs e)

{

IServiceView view = ServiceView.GetInstace((MainView)mainView);

IServiceRepository serviceRepository = new ServiceRepository();

ICareProductRepository careProductRepository = new CareProductRepository();

IPlantRepository plantRepository = new PlantRepository();

IEmployeeRepository employeeRepository = new EmployeeRepository();

new ServicePresenter(view, serviceRepository, careProductRepository, employeeRepository, plantRepository );

}

private void ShowGardenProductView(object? sender, EventArgs e)

{

IGardenView view = GardenView.GetInstace((MainView)mainView);

IGardenRepository gardenRepository = new GardenRepository();

ICityRepository cityRepository = new CityRepository();

IPropertyRepository propertyRepository = new PropertyRepository();

IEmployeeRepository employeeRepository = new EmployeeRepository();

new GardenPresenter(view, gardenRepository, cityRepository, propertyRepository, employeeRepository);

}

private void ShowPlantView(object? sender, EventArgs e)

{

IPlantView view = PlantView.GetInstace((MainView)mainView);

IPlantRepository plantRepository = new PlantRepository();

ICatalogueRepository catalogueRepository = new CatalogueRepository();

new PlantPresenter(view, plantRepository, catalogueRepository);

}

private void ShowEmployeeView(object? sender, EventArgs e)

{

IEmployeeView view = EmployeeView.GetInstace((MainView)mainView);

IEmployeeRepository employeeRepository = new EmployeeRepository();

IPositionRepository positionRepository = new PositionRepository();

new EmployeePresenter(view, employeeRepository, positionRepository);

}

private void ShowCatalogueView(object? sender, EventArgs e)

{

ICatalogueView view = CatalogueView.GetInstace((MainView)mainView);

ICatalogueRepository catalogueRepository = new CatalogueRepository();

IGeneraRepository generaRepository = new GeneraRepository();

new CataloguePresenter(view, catalogueRepository, generaRepository);

}

private void ShowGeneraView(object? sender, EventArgs e)

{

IGeneraView view = GeneraView.GetInstace((MainView)mainView);

IGeneraRepository generaRepository = new GeneraRepository();

IFamilyRepository familyRepository = new FamilyRepository();

new GeneraPresenter(view, generaRepository, familyRepository);

}

//Справочники

private void ShowCareProductView(object? sender, EventArgs e)

{

ICareProductView view = CareProductView.GetInstace((MainView)mainView);

ICareProductRepository repository = new CareProductRepository();

new CareProductPresenter(view, repository);

}

private void ShowPositionView(object? sender, EventArgs e)

{

IPositionView view = PositionView.GetInstace((MainView)mainView);

IPositionRepository repository = new PositionRepository();

new PositionPresenter(view, repository);

}

private void ShowPropertyView(object? sender, EventArgs e)

{

IPropertyView view = PropertyView.GetInstace((MainView)mainView);

IPropertyRepository repository = new PropertyRepository();

new PropertyPresenter(view, repository);

}

private void ShowFamilyView(object? sender, EventArgs e)

{

IFamilyView view = FamilyView.GetInstace((MainView)mainView);

IFamilyRepository repository = new FamilyRepository();

new FamilyPresenter(view, repository);

}

private void ShowCitiesView(object sender, EventArgs e)

{

ICityView view = CityView.GetInstace((MainView)mainView);

ICityRepository repository = new CityRepository();

new CityPresenter(view, repository);

}

}

}

using DataBaseKursRabota.Models;

using DataBaseKursRabota.Repositories;

using DataBaseKursRabota.Views.Tables;

using OfficeOpenXml;

using System.Data;

namespace DataBaseKursRabota.Presenters

{

public class QueryPresenter

{

private IQueryView view;

private IQueryRepository repository;

private BindingSource dataBindingSourse;

private IEnumerable<Dictionary<string, object>> dataList;

public QueryPresenter(IQueryView view, IQueryRepository repository)

{

this.dataBindingSourse = new BindingSource();

this.view = view;

this.repository = repository;

this.view.QueryChangedEvent += ChangeQuery;

this.view.SaveExcelEvent += SaveExcel;

this.view.ShowPlotChartEvent += ShowPlotChart;

this.view.SetDataListBindingSource(dataBindingSourse);

this.view.Show();

}

private void ChangeQuery(object? sender, EventArgs e)

{

if (view.QueryNumber == 18)

{

view.IsChartable = true;

}

if (view.QueryNumber == 0)

{

view.IsExportable = false;

return;

}

view.IsExportable = true;

dataList = repository.Query(view.QueryNumber);

DataTable dt = new DataTable();

foreach (var dict in dataList)

{

DataRow row = dt.NewRow();

foreach (var entry in dict)

{

if (!dt.Columns.Contains(entry.Key))

{

dt.Columns.Add(entry.Key, entry.Value.GetType());

}

row[entry.Key] = entry.Value;

}

dt.Rows.Add(row);

}

dataBindingSourse.DataSource = dt;

}

private void ShowPlotChart(object? sender, EventArgs e)

{

var obj = new ObjToDiagram();

obj.Title = view.QueryNumber.ToString();

var dictionaries = new Dictionary<string, double>();

if(view.QueryNumber == 18)

{

view.IsChartable = true;

foreach (var dictionary in dataList)

{

dictionaries.Add((string)dictionary["Название"], (double)dictionary["Средняя цена"]);

}

}

obj.Values = dictionaries;

var plotView = new PlotView(obj);

plotView.Show();

}

private void SaveExcel(object? sender, EventArgs e)

{

SaveFileDialog dialog = new SaveFileDialog();

dialog.FileName = view.QueryNumber.ToString();

dialog.Filter = "Excel files|\*.xlsx";

var res = dialog.ShowDialog();

if (res != DialogResult.OK)

{

return;

}

ExcelPackage.LicenseContext = LicenseContext.NonCommercial;

using (ExcelPackage package = new ExcelPackage())

{

ExcelWorksheet worksheet = package.Workbook.Worksheets.Add("Sheet1");

DataTable dataTable = ((DataTable)dataBindingSourse.DataSource).Copy();

worksheet.Cells["A1"].LoadFromDataTable(dataTable, true);

using (Stream stream = new FileStream(dialog.FileName, FileMode.Create))

{

package.SaveAs(stream);

}

}

view.Message = "Экспорт выполнен";

}

}

}

using DataBaseKursRabota.Models;

using DataBaseKursRabota.Views;

namespace DataBaseKursRabota.Presenters

{

public class CareProductPresenter

{

private ICareProductView view;

private ICareProductRepository repository;

private BindingSource dataBindingSourse;

private IEnumerable<CareProductModel> dataList;

public CareProductPresenter(ICareProductView view, ICareProductRepository repository)

{

this.dataBindingSourse = new BindingSource();

this.view = view;

this.repository = repository;

this.view.SearchEvent += Search;

this.view.AddNewEvent += AddNew;

this.view.EditEvent += LoadSelectedToEdit;

this.view.DeleteEvent += DeleteSelectedCareProduct;

this.view.NextPageEvent += NextPage;

this.view.PrevPageEvent += PrevPage;

this.view.SaveEvent += SaveCareProduct;

this.view.CancelEvent += CancelAction;

this.view.SetCareProductListBindingSource(dataBindingSourse);

LoadAllCareProductList(view.Page);

this.view.Show();

}

private void PrevPage(object? sender, EventArgs e)

{

view.Page--;

LoadAllCareProductList(view.Page);

}

private void NextPage(object? sender, EventArgs e)

{

view.Page++;

LoadAllCareProductList(view.Page);

}

private void LoadAllCareProductList(int page)

{

var offset = (view.Page - 1) \* 50;

if (offset + 50 < view.Total) view.IsLastPage = true;

else view.IsLastPage = false;

dataList = repository.GetAll(page, 50);

dataBindingSourse.DataSource = dataList;

view.SetTotal(repository.GetTotal());

}

private void CancelAction(object? sender, EventArgs e)

{

CleanViewFields();

}

private void SaveCareProduct(object? sender, EventArgs e)

{

var model = new CareProductModel();

model.Id = Convert.ToInt32(view.CareProductId);

model.CareProduct = view.CareProduct;

model.Unit = view.Unit;

try

{

if (view.IsEdit)

{

repository.Edit(model);

view.Message = "Средство ухода успешно изменен";

}

else

{

repository.Add(model);

view.Message = "Средство ухода успешно добавлен";

}

view.IsSuccessful = true;

LoadAllCareProductList(view.Page);

CleanViewFields();

}

catch (Exception ex)

{

view.IsSuccessful = false;

view.Message = ex.Message;

}

}

private void CleanViewFields()

{

view.CareProductId = "0";

view.CareProduct = "";

view.Unit = "";

}

private void DeleteSelectedCareProduct(object? sender, EventArgs e)

{

try

{

var model = (CareProductModel)dataBindingSourse.Current;

repository.Delete(model.Id);

view.IsSuccessful = true;

view.Message = "Средство ухода успешно удален";

LoadAllCareProductList(view.Page);

}

catch (Exception ex)

{

view.IsSuccessful = false;

view.Message = ex.Message;

};

}

private void LoadSelectedToEdit(object? sender, EventArgs e)

{

var model = (CareProductModel)dataBindingSourse.Current;

view.CareProductId = model.Id.ToString();

view.CareProduct = model.CareProduct;

view.Unit = model.Unit;

view.IsEdit = true;

}

private void AddNew(object? sender, EventArgs e)

{

view.IsEdit = false;

}

private void Search(object? sender, EventArgs e)

{

bool emptyValue = string.IsNullOrEmpty(this.view.SearchValue);

if (emptyValue == false)

{

dataList = repository.GetByValue(this.view.SearchValue);

}

else

{

dataList = repository.GetAll(view.Page, 50);

}

dataBindingSourse.DataSource = dataList;

}

}

}

using DataBaseKursRabota.Views;

using DataBaseKursRabota.Models;

namespace DataBaseKursRabota.Presenters

{

public class CityPresenter

{

private ICityView view;

private ICityRepository repository;

private BindingSource dataBindingSourse;

private IEnumerable<CityModel> dataList;

public CityPresenter(ICityView view, ICityRepository repository)

{

this.dataBindingSourse = new BindingSource();

this.view = view;

this.repository = repository;

this.view.SearchEvent += Search;

this.view.AddNewEvent += AddNew;

this.view.EditEvent += LoadSelectedToEdit;

this.view.DeleteEvent += DeleteSelectedCity;

this.view.NextPageEvent += NextPage;

this.view.PrevPageEvent += PrevPage;

this.view.SaveEvent += SaveCity;

this.view.CancelEvent += CancelAction;

this.view.SetCityListBindingSource(dataBindingSourse);

LoadAllCityList(view.Page);

this.view.Show();

}

private void PrevPage(object? sender, EventArgs e)

{

view.Page--;

LoadAllCityList(view.Page);

}

private void NextPage(object? sender, EventArgs e)

{

view.Page++;

LoadAllCityList(view.Page);

}

private void LoadAllCityList(int page)

{

var offset = (view.Page - 1) \* 50;

if (offset + 50 < view.Total) view.IsLastPage = true;

else view.IsLastPage = false;

dataList = repository.GetAll(page, 50);

dataBindingSourse.DataSource = dataList;

view.SetTotal(repository.GetTotal());

}

private void CancelAction(object? sender, EventArgs e)

{

CleanViewFields();

}

private void SaveCity(object? sender, EventArgs e)

{

var model = new CityModel();

model.Id = Convert.ToInt32(view.CityId);

model.City = view.City;

try

{

if (view.IsEdit)

{

repository.Edit(model);

view.Message = "Город успешно изменен";

}

else

{

repository.Add(model);

view.Message = "Город успешно добавлен";

}

view.IsSuccessful = true;

LoadAllCityList(view.Page);

CleanViewFields();

}

catch(Exception ex)

{

view.IsSuccessful= false;

view.Message = ex.Message;

}

}

private void CleanViewFields()

{

view.CityId = "0";

view.City = "";

}

private void DeleteSelectedCity(object? sender, EventArgs e)

{

try

{

var city = (CityModel)dataBindingSourse.Current;

repository.Delete(city.Id);

view.IsSuccessful = true;

view.Message = "Город успешно удален";

LoadAllCityList(view.Page);

}

catch (Exception ex)

{

view.IsSuccessful = false;

view.Message = ex.Message;

};

}

private void LoadSelectedToEdit(object? sender, EventArgs e)

{

var city = (CityModel)dataBindingSourse.Current;

view.CityId = city.Id.ToString();

view.City = city.City;

view.IsEdit = true;

}

private void AddNew(object? sender, EventArgs e)

{

view.IsEdit = false;

}

private void Search(object? sender, EventArgs e)

{

bool emptyValue = string.IsNullOrEmpty(this.view.SearchValue);

if(emptyValue == false)

{

dataList = repository.GetByValue(this.view.SearchValue);

}

else

{

dataList = repository.GetAll(view.Page, 50);

}

dataBindingSourse.DataSource = dataList;

}

}

}

using DataBaseKursRabota.Models;

using DataBaseKursRabota.Views;

namespace DataBaseKursRabota.Presenters

{

public class FamilyPresenter

{

private IFamilyView view;

private IFamilyRepository repository;

private BindingSource dataBindingSourse;

private IEnumerable<FamilyModel> dataList;

public FamilyPresenter(IFamilyView view, IFamilyRepository repository)

{

this.dataBindingSourse = new BindingSource();

this.view = view;

this.repository = repository;

this.view.SearchEvent += Search;

this.view.AddNewEvent += AddNew;

this.view.EditEvent += LoadSelectedToEdit;

this.view.DeleteEvent += DeleteSelectedFamily;

this.view.NextPageEvent += NextPage;

this.view.PrevPageEvent += PrevPage;

this.view.SaveEvent += SaveFamily;

this.view.CancelEvent += CancelAction;

this.view.SetFamilyListBindingSource(dataBindingSourse);

LoadAllFamilyList(view.Page);

this.view.Show();

}

private void PrevPage(object? sender, EventArgs e)

{

view.Page--;

LoadAllFamilyList(view.Page);

}

private void NextPage(object? sender, EventArgs e)

{

view.Page++;

LoadAllFamilyList(view.Page);

}

private void LoadAllFamilyList(int page)

{

var offset = (view.Page - 1) \* 50;

if (offset + 50 < view.Total) view.IsLastPage = true;

else view.IsLastPage = false;

dataList = repository.GetAll(page, 50);

dataBindingSourse.DataSource = dataList;

view.SetTotal(repository.GetTotal());

}

private void CancelAction(object? sender, EventArgs e)

{

CleanViewFields();

}

private void SaveFamily(object? sender, EventArgs e)

{

var model = new FamilyModel();

model.Id = Convert.ToInt32(view.FamilyId);

model.Family = view.Family;

try

{

if (view.IsEdit)

{

repository.Edit(model);

view.Message = "Семейство успешно изменен";

}

else

{

repository.Add(model);

view.Message = "Семейство успешно добавлен";

}

view.IsSuccessful = true;

LoadAllFamilyList(view.Page);

CleanViewFields();

}

catch (Exception ex)

{

view.IsSuccessful = false;

view.Message = ex.Message;

}

}

private void CleanViewFields()

{

view.FamilyId = "0";

view.Family = "";

}

private void DeleteSelectedFamily(object? sender, EventArgs e)

{

try

{

var family = (FamilyModel)dataBindingSourse.Current;

repository.Delete(family.Id);

view.IsSuccessful = true;

view.Message = "Семейство успешно удален";

LoadAllFamilyList(view.Page);

}

catch (Exception ex)

{

view.IsSuccessful = false;

view.Message = ex.Message;

};

}

private void LoadSelectedToEdit(object? sender, EventArgs e)

{

var family = (FamilyModel)dataBindingSourse.Current;

view.FamilyId = family.Id.ToString();

view.Family = family.Family;

view.IsEdit = true;

}

private void AddNew(object? sender, EventArgs e)

{

view.IsEdit = false;

}

private void Search(object? sender, EventArgs e)

{

bool emptyValue = string.IsNullOrEmpty(this.view.SearchValue);

if (emptyValue == false)

{

dataList = repository.GetByValue(this.view.SearchValue);

}

else

{

dataList = repository.GetAll(view.Page, 50);

}

dataBindingSourse.DataSource = dataList;

}

}

}

using DataBaseKursRabota.Models;

using DataBaseKursRabota.Views;

namespace DataBaseKursRabota.Presenters

{

public class PositionPresenter

{

private IPositionView view;

private IPositionRepository repository;

private BindingSource dataBindingSourse;

private IEnumerable<PositionModel> dataList;

public PositionPresenter(IPositionView view, IPositionRepository repository)

{

this.dataBindingSourse = new BindingSource();

this.view = view;

this.repository = repository;

this.view.SearchEvent += Search;

this.view.AddNewEvent += AddNew;

this.view.EditEvent += LoadSelectedToEdit;

this.view.DeleteEvent += DeleteSelectedPosition;

this.view.NextPageEvent += NextPage;

this.view.PrevPageEvent += PrevPage;

this.view.SaveEvent += SavePosition;

this.view.CancelEvent += CancelAction;

this.view.SetPositionListBindingSource(dataBindingSourse);

LoadAllPositionList(view.Page);

this.view.Show();

}

private void PrevPage(object? sender, EventArgs e)

{

view.Page--;

LoadAllPositionList(view.Page);

}

private void NextPage(object? sender, EventArgs e)

{

view.Page++;

LoadAllPositionList(view.Page);

}

private void LoadAllPositionList(int page)

{

var offset = (view.Page - 1) \* 50;

if (offset + 50 < view.Total) view.IsLastPage = true;

else view.IsLastPage = false;

dataList = repository.GetAll(page, 50);

dataBindingSourse.DataSource = dataList;

view.SetTotal(repository.GetTotal());

}

private void CancelAction(object? sender, EventArgs e)

{

CleanViewFields();

}

private void SavePosition(object? sender, EventArgs e)

{

var model = new PositionModel();

model.Id = Convert.ToInt32(view.PositionId);

model.Position = view.Position;

try

{

if (view.IsEdit)

{

repository.Edit(model);

view.Message = "Должность успешно изменен";

}

else

{

repository.Add(model);

view.Message = "Должность успешно добавлен";

}

view.IsSuccessful = true;

LoadAllPositionList(view.Page);

CleanViewFields();

}

catch (Exception ex)

{

view.IsSuccessful = false;

view.Message = ex.Message;

}

}

private void CleanViewFields()

{

view.PositionId = "0";

view.Position = "";

}

private void DeleteSelectedPosition(object? sender, EventArgs e)

{

try

{

var city = (PositionModel)dataBindingSourse.Current;

repository.Delete(city.Id);

view.IsSuccessful = true;

view.Message = "Должность успешно удален";

LoadAllPositionList(view.Page);

}

catch (Exception ex)

{

view.IsSuccessful = false;

view.Message = ex.Message;

};

}

private void LoadSelectedToEdit(object? sender, EventArgs e)

{

var city = (PositionModel)dataBindingSourse.Current;

view.PositionId = city.Id.ToString();

view.Position = city.Position;

view.IsEdit = true;

}

private void AddNew(object? sender, EventArgs e)

{

view.IsEdit = false;

}

private void Search(object? sender, EventArgs e)

{

bool emptyValue = string.IsNullOrEmpty(this.view.SearchValue);

if (emptyValue == false)

{

dataList = repository.GetByValue(this.view.SearchValue);

}

else

{

dataList = repository.GetAll(view.Page, 50);

}

dataBindingSourse.DataSource = dataList;

}

}

}

using DataBaseKursRabota.Models;

using DataBaseKursRabota.Views;

namespace DataBaseKursRabota.Presenters

{

public class PropertyPresenter

{

private IPropertyView view;

private IPropertyRepository repository;

private BindingSource dataBindingSourse;

private IEnumerable<PropertyModel> dataList;

public PropertyPresenter(IPropertyView view, IPropertyRepository repository)

{

this.dataBindingSourse = new BindingSource();

this.view = view;

this.repository = repository;

this.view.SearchEvent += Search;

this.view.AddNewEvent += AddNew;

this.view.EditEvent += LoadSelectedToEdit;

this.view.DeleteEvent += DeleteSelectedProperty;

this.view.NextPageEvent += NextPage;

this.view.PrevPageEvent += PrevPage;

this.view.SaveEvent += SaveProperty;

this.view.CancelEvent += CancelAction;

this.view.SetPropertyListBindingSource(dataBindingSourse);

LoadAllPropertyList(view.Page);

this.view.Show();

}

private void PrevPage(object? sender, EventArgs e)

{

view.Page--;

LoadAllPropertyList(view.Page);

}

private void NextPage(object? sender, EventArgs e)

{

view.Page++;

LoadAllPropertyList(view.Page);

}

private void LoadAllPropertyList(int page)

{

var offset = (view.Page - 1) \* 50;

if (offset + 50 < view.Total) view.IsLastPage = true;

else view.IsLastPage = false;

dataList = repository.GetAll(page, 50);

dataBindingSourse.DataSource = dataList;

view.SetTotal(repository.GetTotal());

}

private void CancelAction(object? sender, EventArgs e)

{

CleanViewFields();

}

private void SaveProperty(object? sender, EventArgs e)

{

var model = new PropertyModel();

model.Id = Convert.ToInt32(view.PropertyId);

model.Property = view.Property;

try

{

if (view.IsEdit)

{

repository.Edit(model);

view.Message = "Тип собственности успешно изменен";

}

else

{

repository.Add(model);

view.Message = "Тип собственности успешно добавлен";

}

view.IsSuccessful = true;

LoadAllPropertyList(view.Page);

CleanViewFields();

}

catch (Exception ex)

{

view.IsSuccessful = false;

view.Message = ex.Message;

}

}

private void CleanViewFields()

{

view.PropertyId = "0";

view.Property = "";

}

private void DeleteSelectedProperty(object? sender, EventArgs e)

{

try

{

var city = (PropertyModel)dataBindingSourse.Current;

repository.Delete(city.Id);

view.IsSuccessful = true;

view.Message = "Тип собственности успешно удален";

LoadAllPropertyList(view.Page);

}

catch (Exception ex)

{

view.IsSuccessful = false;

view.Message = ex.Message;

};

}

private void LoadSelectedToEdit(object? sender, EventArgs e)

{

var city = (PropertyModel)dataBindingSourse.Current;

view.PropertyId = city.Id.ToString();

view.Property = city.Property;

view.IsEdit = true;

}

private void AddNew(object? sender, EventArgs e)

{

view.IsEdit = false;

}

private void Search(object? sender, EventArgs e)

{

bool emptyValue = string.IsNullOrEmpty(this.view.SearchValue);

if (emptyValue == false)

{

dataList = repository.GetByValue(this.view.SearchValue);

}

else

{

dataList = repository.GetAll(view.Page, 50);

}

dataBindingSourse.DataSource = dataList;

}

}

}

using AutoMapper;

using DataBaseKursRabota.Models.Tables;

using DataBaseKursRabota.Models;

using DataBaseKursRabota.Views.Tables;

using DataBaseKursRabota.Views.ViewModels.Tables;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataBaseKursRabota.Presenters.Tables

{

public class CataloguePresenter

{

private ICatalogueView view;

private ICatalogueRepository catalogueRepository;

private IGeneraRepository generaRepository;

private BindingSource cataloguesBindingSourse;

private BindingSource generaBindingSourse;

private IEnumerable<GeneraDataModel> generalist;

private IEnumerable<CatalogueDataModel> catalogueList;

public CataloguePresenter(ICatalogueView view, ICatalogueRepository catalogueRepository, IGeneraRepository generaRepository)

{

this.cataloguesBindingSourse = new BindingSource();

this.generaBindingSourse = new BindingSource();

this.view = view;

this.catalogueRepository = catalogueRepository;

this.generaRepository = generaRepository;

this.view.SearchEvent += SearchCatalogue;

this.view.AddNewEvent += AddNewCatalogue;

this.view.EditEvent += LoadSelectedCatalogueToEdit;

this.view.DeleteEvent += DeleteSelectedCatalogue;

this.view.NextPageEvent += NextPage;

this.view.PrevPageEvent += PrevPage;

this.view.SaveEvent += SaveCatalogue;

this.view.CancelEvent += CancelAction;

this.view.SetCatalogueListBindingSource(cataloguesBindingSourse);

this.view.SetGeneraListComboBox(generaBindingSourse);

LoadAllGeneraList();

LoadAllCatalogueList(view.Page);

this.view.Show();

}

private void PrevPage(object? sender, EventArgs e)

{

view.Page--;

LoadAllCatalogueList(view.Page);

}

private void NextPage(object? sender, EventArgs e)

{

view.Page++;

LoadAllCatalogueList(view.Page);

}

private void LoadAllGeneraList()

{

generalist = generaRepository.GetAll(1, 999999);

generaBindingSourse.DataSource = generalist;

}

private void LoadAllCatalogueList(int page)

{

var offset = (view.Page - 1) \* 50;

if (offset + 50 < view.Total) view.IsLastPage = true;

else view.IsLastPage = false;

catalogueList = catalogueRepository.GetAll(page, 50);

var config = new MapperConfiguration(cfg => cfg.CreateMap<CatalogueDataModel, CatalogueViewModel>());

var mapper = config.CreateMapper();

IEnumerable<CatalogueViewModel> viewModels = mapper.Map<List<CatalogueViewModel>>(catalogueList);

cataloguesBindingSourse.DataSource = viewModels;

view.SetTotal(catalogueRepository.GetTotal());

}

private void CancelAction(object? sender, EventArgs e)

{

CleanViewFields();

}

private void SaveCatalogue(object? sender, EventArgs e)

{

var model = new CatalogueDataModel();

model.Id = Convert.ToInt32(view.CatalogueId);

model.Fullname = view.FullName;

model.LifeSpan = view.LifeSpan;

model.Annual = view.Annual;

model.GeneraId = Convert.ToInt32(view.GeneraId);

try

{

if (view.IsEdit)

{

catalogueRepository.Edit(model);

view.Message = "Вид успешно изменен";

}

else

{

catalogueRepository.Add(model);

view.Message = "Вид успешно добавлен";

}

view.IsSuccessful = true;

LoadAllCatalogueList(view.Page);

CleanViewFields();

}

catch (Exception ex)

{

view.IsSuccessful = false;

view.Message = ex.Message;

}

}

private void CleanViewFields()

{

view.CatalogueId = "0";

view.FullName = "";

view.LifeSpan = 1;

}

private void DeleteSelectedCatalogue(object? sender, EventArgs e)

{

try

{

var catalogue = (CatalogueViewModel)cataloguesBindingSourse.Current;

catalogueRepository.Delete(catalogue.Id);

view.IsSuccessful = true;

view.Message = "Вид успешно удален";

LoadAllCatalogueList(view.Page);

}

catch (Exception ex)

{

view.IsSuccessful = false;

view.Message = ex.Message;

};

}

private void LoadSelectedCatalogueToEdit(object? sender, EventArgs e)

{

var catalogue = (CatalogueViewModel)cataloguesBindingSourse.Current;

view.CatalogueId = catalogue.Id.ToString();

view.FullName = catalogue.Fullname;

view.Genera = catalogue.Genera;

view.LifeSpan = catalogue.LifeSpan;

view.IsEdit = true;

}

private void AddNewCatalogue(object? sender, EventArgs e)

{

view.IsEdit = false;

}

private void SearchCatalogue(object? sender, EventArgs e)

{

bool emptyValue = string.IsNullOrEmpty(this.view.SearchValue);

if (emptyValue == false)

{

catalogueList = catalogueRepository.GetByValue(this.view.SearchValue);

}

else

{

catalogueList = catalogueRepository.GetAll(view.Page, 50);

}

cataloguesBindingSourse.DataSource = catalogueList;

}

}

}

using AutoMapper;

using DataBaseKursRabota.Models;

using DataBaseKursRabota.Models.Tables;

using DataBaseKursRabota.Views.Tables;

using DataBaseKursRabota.Views.ViewModels.Tables;

namespace DataBaseKursRabota.Presenters.Tables

{

public class EmployeePresenter

{

private IEmployeeView view;

private IEmployeeRepository employeeRepository;

private IPositionRepository positionRepository;

private BindingSource employeesBindingSourse;

private BindingSource positionBindingSourse;

private IEnumerable<PositionModel> positionlist;

private IEnumerable<EmployeeDataModel> employeeList;

public EmployeePresenter(IEmployeeView view, IEmployeeRepository employeeRepository, IPositionRepository positionRepository)

{

this.employeesBindingSourse = new BindingSource();

this.positionBindingSourse = new BindingSource();

this.view = view;

this.employeeRepository = employeeRepository;

this.positionRepository = positionRepository;

this.view.SearchEvent += SearchEmployee;

this.view.AddNewEvent += AddNewEmployee;

this.view.EditEvent += LoadSelectedEmployeeToEdit;

this.view.DeleteEvent += DeleteSelectedEmployee;

this.view.NextPageEvent += NextPage;

this.view.PrevPageEvent += PrevPage;

this.view.SaveEvent += SaveEmployee;

this.view.CancelEvent += CancelAction;

this.view.StartGenerateEvent += GenerateRows;

this.view.CancelGenerateEvent += CancelGenerateRows;

this.view.SetEmployeeListBindingSource(employeesBindingSourse);

this.view.SetPositionListComboBox(positionBindingSourse);

LoadAllPositionList();

LoadAllEmployeeList(view.Page);

this.view.Show();

}

private void CancelGenerateRows(object? sender, EventArgs e)

{

view.IsGenerating = false;

view.Message = "Записи сгенерированы.";

LoadAllEmployeeList(view.Page);

}

private async void GenerateRows(object? sender, EventArgs e)

{

view.IsGenerating = true;

for (int i = 0; i < 10000; i++)

{

if (!view.IsGenerating) break;

EmployeeDataModel employee = employeeRepository.Generate(positionlist);

await employeeRepository.AddEmployeeAsync(employee);

if(i%10 == 0) view.SetTotal(employeeRepository.GetTotal());

}

view.Message = "Записи сгенерированы.";

LoadAllEmployeeList(view.Page);

}

private void LoadAllPositionList()

{

positionlist = positionRepository.GetAll(1, 999999);

positionBindingSourse.DataSource = positionlist;

}

private void LoadAllEmployeeList(int page)

{

var offset = (view.Page - 1) \* 50;

if (offset + 50 < view.Total) view.IsLastPage = true;

else view.IsLastPage = false;

employeeList = employeeRepository.GetAll(page, 50);

var config = new MapperConfiguration(cfg => cfg.CreateMap<EmployeeDataModel, EmployeeViewModel>());

var mapper = config.CreateMapper();

IEnumerable<EmployeeViewModel> viewModels = mapper.Map<List<EmployeeViewModel>>(employeeList);

employeesBindingSourse.DataSource = viewModels;

view.SetTotal(employeeRepository.GetTotal());

}

private void CancelAction(object? sender, EventArgs e)

{

CleanViewFields();

}

private void SaveEmployee(object? sender, EventArgs e)

{

var model = new EmployeeDataModel();

model.Id = Convert.ToInt32(view.EmployeeId);

model.Surname = view.Surname;

model.Name = view.Name;

model.Patronymic = view.Patronymic;

model.Birth = view.Birth;

model.Experience = view.Experience;

model.Salary = view.Salary;

model.PositionId = Convert.ToInt32(view.PositionId);

try

{

if (view.IsEdit)

{

employeeRepository.Edit(model);

view.Message = "Работник успешно изменен";

}

else

{

employeeRepository.Add(model);

view.Message = "Работник успешно добавлен";

}

view.IsSuccessful = true;

LoadAllEmployeeList(view.Page);

CleanViewFields();

}

catch (Exception ex)

{

view.IsSuccessful = false;

view.Message = ex.Message;

}

}

private void CleanViewFields()

{

view.EmployeeId = "0";

view.Surname = "";

view.Name = "";

view.Birth = "";

view.Salary = 1;

view.Experience = 1;

}

private void DeleteSelectedEmployee(object? sender, EventArgs e)

{

try

{

var employee = (EmployeeViewModel)employeesBindingSourse.Current;

employeeRepository.Delete(employee.Id);

view.IsSuccessful = true;

view.Message = "Работник успешно удален";

LoadAllEmployeeList(view.Page);

}

catch (Exception ex)

{

view.IsSuccessful = false;

view.Message = ex.Message;

};

}

private void LoadSelectedEmployeeToEdit(object? sender, EventArgs e)

{

var employee = (EmployeeViewModel)employeesBindingSourse.Current;

view.EmployeeId = employee.Id.ToString();

view.Surname = employee.Surname;

view.Name = employee.Name;

view.Patronymic = employee.Patronymic;

view.Birth = employee.Birth;

view.Salary = employee.Salary;

view.Experience = employee.Experience;

view.Position = employee.Position;

view.IsEdit = true;

}

private void AddNewEmployee(object? sender, EventArgs e)

{

view.IsEdit = false;

}

private void SearchEmployee(object? sender, EventArgs e)

{

bool emptyValue = string.IsNullOrEmpty(this.view.SearchValue);

if (emptyValue == false)

{

employeeList = employeeRepository.GetByValue(this.view.SearchValue);

}

else

{

employeeList = employeeRepository.GetAll(view.Page, 50);

}

employeesBindingSourse.DataSource = employeeList;

}

private void PrevPage(object? sender, EventArgs e)

{

view.Page--;

LoadAllEmployeeList(view.Page);

}

private void NextPage(object? sender, EventArgs e)

{

view.Page++;

ПРИЛОЖЕНИЕ В.  
РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

В1. Назначение программы

Программа предназначена для учета деятельности ботанических садов страны с целью оптимизации работы сотрудников. Приложение позволяет пользователю добавлять данные в таблицу, редактировать данные и удалять данные из таблиц. Также пользователю доступны функции поиск и фильтр для быстрого доступа к требующимся данным, генерация записей в таблицах и сохранение результатов некоторых запросов в форме отчета.

В2. Условия выполнения программы

Для запуска программного продукта необходима операционная система MS Windows (версия не ниже XP). Объем свободной памяти на жестком диске не менее 600 Мбайт. Процессор не менее производительный, чем Intel Pentium 4 с тактовой частотой 3,00 ГГц.

В3. Выполнение программы

Для работы с программой необходимо предварительно авторизоваться. Форма авторизации изображена на рисунке В.1

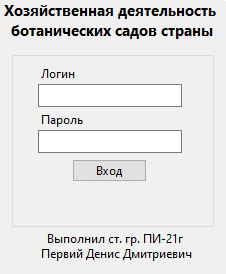


Рисунок В.1 – Форма авторизации

После удачной авторизации перед пользователем появляется основная форма. На основной форме слева находится меню (см. рис. В.2) выбора таблиц, справочников и запросов.

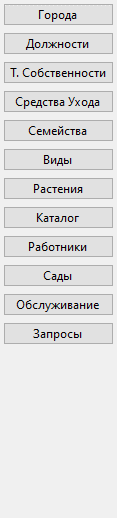


Рисунок В.2 – Меню выбора таблиц

У каждой таблицы есть кнопки базового взаимодействия с таблицей (см. рис. В.3) «Добавить», «Редактировать» и «Удалить», а также и переключение между страницами.



Рисунок В.3 – Меню взаимодействия с таблицей

По нажатию на кнопку «Добавить» открывается форма добавления записи в выбранную таблицу (см. рис. В.4)

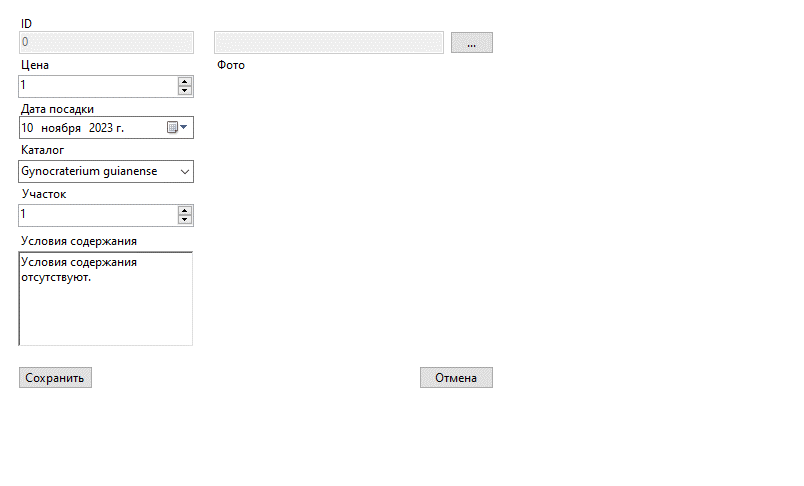


Рисунок В.4 – Пример форма добавления записи в таблицу

По нажатию кнопки «Редактировать» открывается форма редактирования похожая на форму добавления с одним отличием, что поля для ввода уже заполнены информацией, выбранной записи, при изменении которых и сохранений новые данные отправляются в базу данных.

При нажатии кнопки «Удалить» открывается диалоговое окно с подтверждением решения удаления записи. (см. рис. В.5)

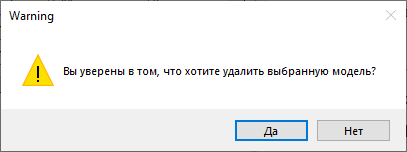


Рисунок В.5 – Диалоговое окно подтверждения удаления записи

У каждой таблицы есть панель для поиска записей, состоящая из поля для ввода текста запроса, выбора столбца, по которому будет произведён поиск, кнопки «Поиск», которая будет воспроизводить поиск. На рисунке В.6 изображено поле для поиска



Рисунок В.6 – Панель для поиска записей

По нажатию на кнопку «Запроса» открывается новое окно выбора запроса. Окно содержит выбор запроса и кнопки экспорта в Excel и диаграмму. На рисунке В.7 изображён пример такой формы.

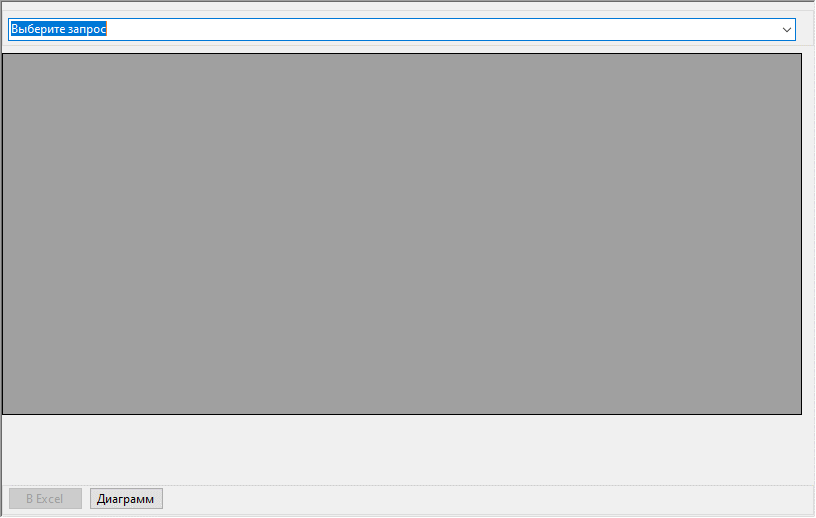


Рисунок В.7 – Форма «Выбор запроса»