|  |  |
| --- | --- |
| МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  РОССИЙСКОЙ ФЕДЕРАЦИИ  Федеральное государственное автономное образовательное учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | |
| *Кафедра математического обеспечения  вычислительных систем* | |
|  | |
| ОТЧЕТ  по заданию № 7-8  по дисциплине Учебная практика по базам данных и СУБД  «Объектно-реляционное отображение и Автоматизированное рабочее место» | |
|  | Работу выполнила студент группы *ПМИ-1* *3* курса механико-математического факультета  Жуков А.Е.  4 января 2022 г.  Принял:  ассистент каф. МОВС  Турова И.А.  “\_\_\_\_\_” 2021 г. |
| Пермь 2021 | |

# Постановка задачи

Написать программу, которая на графическом интерфейсе пользователя позволяет просматривать, добавлять, удалять, изменять записи в двух других связанных таблицах из спроектированной ранее базы данных предметной области, без необходимости задания пользователем и отображения ему искусственных ключей. На уровне программного кода CRUD-операции должны выполняться с использованием некоторой ORM-библиотеки.

Провести тестирование корректности работы программы.

Написать отчёт о проделанной работе.

Написать программу, которая на графическом интерфейсе пользователя позволяет просматривать, добавлять, удалять, изменять записи и осуществлять поиск минимум двух стандартных запросов в 4 связанных таблицах из спроектированной ранее базы данных предметной области без необходимости задания и отображения пользователю искусственных ключей. Между указанными таблицами должно присутствовать как отношение типа 1:М, так и отношение типа М:М.

При запуске программы пользователю должна отображаться форма аутентификации по логину и паролю, допускающая гостевой вход. Из формы аутентификации должна быть возможность перейти на форму регистрации нового логина и пароля для оператора АРМ.

Пользователь, зашедший в приложение под учётной записью администратора АРМ, должен иметь возможность просматривать, добавлять, удалять, изменять записи во всех используемых таблицах из спроектированной ранее базы данных предметной области и в таблице учётных записей пользователей. Пользователь, зашедший в приложение под учётной записью оператора АРМ, должен иметь возможность просматривать, добавлять, удалять, изменять записи только во всех используемых таблицах из спроектированной ранее базы данных предметной области. Пользователь, зашедший в приложение под гостевой учётной записью АРМ, должен иметь возможность только просматривать записи во всех используемых таблицах из спроектированной ранее базы данных предметной области.

Средства стандартного поиска доступны всем категориям пользователей.

Провести тестирование корректности работы программы.

Написать отчёт о проделанной работе.

# Анализ предметной области

Для решения поставленной задачи из предметной области «mri machines» (аппараты МРТ) были выбраны таблицы: «buildings», «clinics», «employers», «position». Их структура следующая:

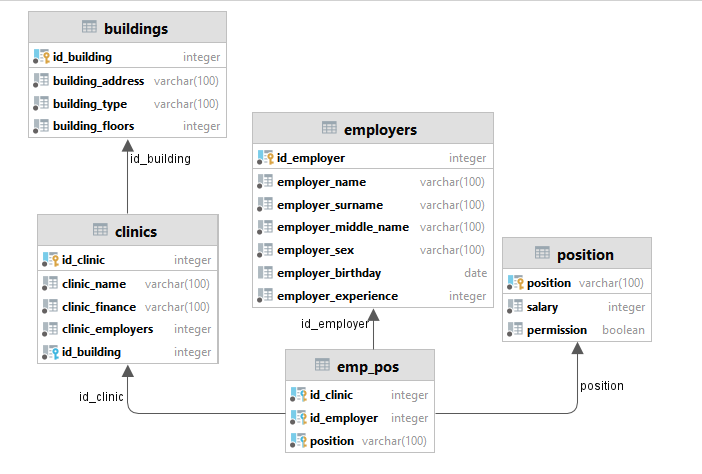


Рис 1 – Структура таблиц

Так же, для реализации работы с пользователями была создана таблица «users»:

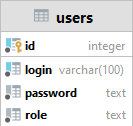


Рис 2 – Структура таблицы users

# Описание функционала разработанной программы

## Внешний вид и функционал приложения

При запуске программы открывается форма входа, откуда доступны авторизация, регистрация и возможность гостевого входа:

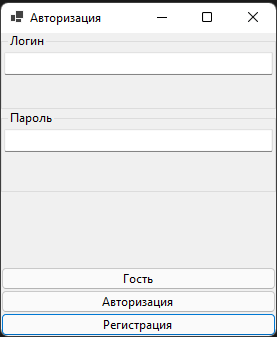


Рис 3 – Форма авторизации

После авторизации открывается главная форма приложения, на которой две таблицы: здания и клиники, в данном здании. Так же есть возможность перейти к таблицам с сотрудниками, должностями и таблице пользователей; выполнить два стандартных запроса поиска; трудоустроить сотрудника.

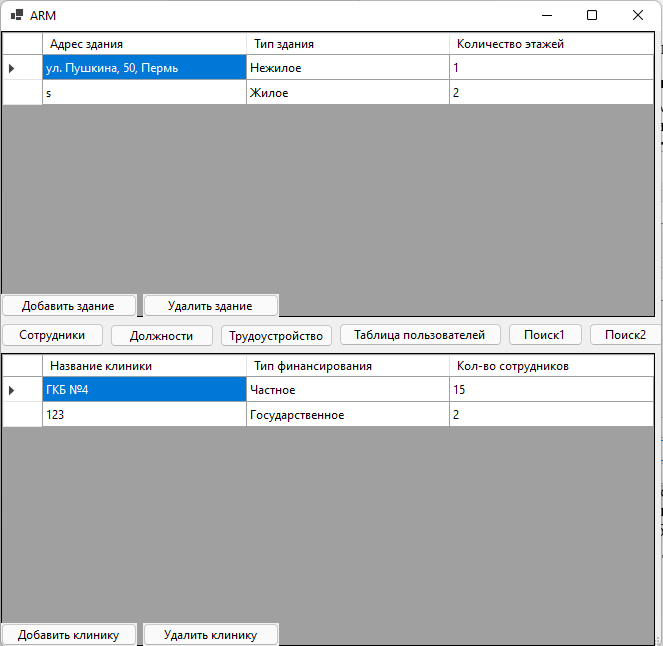


Рис 4 – Главная форма приложения

С главной формы можно перейти к редактированию, добавлению и удалению зданий или клиник.

Большинство форм аналогичны, приведем пример формы с сотрудниками, рассмотрим форму добавления сотрудника.

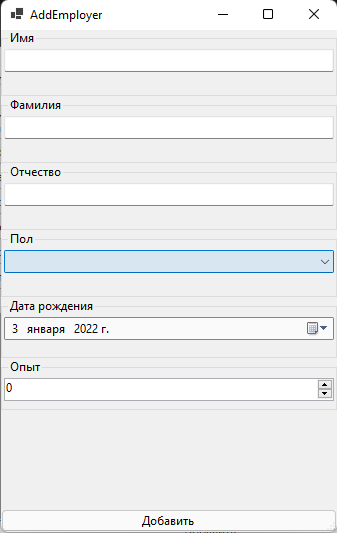


Рис 5 – Форма добавления сотрудника

## Основные алгоритмы работы приложения

1. Редактирование строк
   1. Изменить какой-либо столбец
   2. Проверить измененные данные на допустимость
   3. Обновить данные в базе и вывести обновленные данные на экран
2. Добавление строк
   1. Перейти на форму добавления объекта
   2. Ввести данные
   3. Проверить данные на допустимость
   4. Добавить данные в базу и вывести на экран обновленную таблицу
3. Удаление строк
   1. Выбрать строку
   2. Нажать delete
   3. Удалить данные из базы и вывести на экран обновленную таблицу
4. Регистрация
   1. Проверить, есть ли логин в базе
   2. Проверить пароль на сложность
5. Авторизация
   1. Найти в базе запись с введенным логином
   2. Сверить хеш введенного пароля и хеш из базы
   3. Открыть главную форму, учитывая роль, под которой зашел пользователь

## Реализация работы приложения

Для работы с базой данных используется среда проектирования баз данных PostgreSQL. Для проектирования приложения используется среда разработки Microsoft Visual Studio 2019.

Авторизация и регистрация взяты из лабораторной работы №2 по данной дисциплине, добавилась лишь кнопка гостевого входа, остальное совпадает. При успешном входе открывается главная форма, в конструктор которой передается строка с ролью пользователя. В зависимости от роли блокируются кнопки и свойства таблиц.

**private** **void** **MuteButtons**(**string** role)

{

btn\_addBui.Enabled = **true**;

btn\_delBuil.Enabled = **true**;

btn\_addClinic.Enabled = **true**;

btn\_delClinic.Enabled = **true**;

btn\_openUsers.Enabled = **true**;

**if**(role=="Оператор")

{

btn\_openUsers.Enabled = **false**;

}

**if** (role == "Гость")

{

btn\_addBui.Enabled = **false**;

btn\_delBuil.Enabled = **false**;

btn\_addClinic.Enabled = **false**;

btn\_delClinic.Enabled = **false**;

btn\_openUsers.Enabled = **false**;

dgv\_buildings.ReadOnly = **true**;

dgv\_clinics.ReadOnly = **true**;

}

}

Так же роль передается в формы с сотрудниками, должностями и трудоустройством, где аналогично блокируются любые возможности изменить данные в зависимости от роли.

Чтобы было проще работать, используется ORM, позволяющее работать с базой данных через концепции ООП. Чтобы создать модель нашей базы данных необходимо установить несколько пакетов. Список пакетов, которые использовались при решении поставленной задачи:

1. Npgsql.EntityFrameworkCore.PostgreSQL
2. Npgsql
3. Microsoft.EntityFrameworkCore.Tools
4. Microsoft.EntityFrameworkCore
5. BCrypt.Net-Next (необходим для хеширования паролей)

Далее, необходимо ввести следующую команду в диспетчер пакетов:

Scaffold-DbContext "Host=localhost;Port=5432;Database=mri;Username=postgres;Password=password"

И название провайдера:

Npgsql.EntityFrameworkCore.PostgreSQL

После чего в проекте создадутся классы всех таблиц базы данных. Теперь мы можем работать с этими объектами, а не напрямую с таблицами базы данных.

Чтобы обратиться к базе данных, необходимо создавать контекст. Языком запроса является LINQ. Например, вывод данных в таблицу зданий выглядит следующим образом:

**using** (**var** db = **new** mriContext())

{

**var** result = (**from** Building **in** db.Buildings **select** Building).ToList();

dgv\_buildings.DataSource = result;

dgv\_buildings.Columns["IdBuilding"].Visible = **false**;

dgv\_buildings.Columns["BuildingAddress"].HeaderText = "Адрес здания";

dgv\_buildings.Columns["BuildingType"].HeaderText = "Тип здания";

dgv\_buildings.Columns["BuildingFloors"].HeaderText = "Количество этажей";

dgv\_buildings.Columns["Clinics"].Visible = **false**;

}

По факту, получение данных занимает лишь две строки, что очень кратко и удобно. Далее привязываем наш список зданий в таблице и настраиваем столбцы, скрывая некоторые данные и меняя имена столбцов.

Добавление здания при использовании ORM выглядит следующим образом: создается новый объект здания и заполняются поля данными из формы, вызывается метод Add, куда передается созданный объект, сохраняются изменения.

**using** (**var** db=**new** mriContext())

{

Building building = **new** Building

{

BuildingAddress = tb\_address.Text.ToString(),

BuildingType = cb\_builType.Text.ToString(),

BuildingFloors = (**int**)num\_builFloor.Value

};

db.Add(building);

db.SaveChanges();

}

Редактирование и удаление происходит аналогичным образом, но вызываются соответствующие методы:

**var** id\_del\_build = (**int?**)dgv\_buildings.CurrentRow.Index;

**if** (id\_del\_build.HasValue)

{

**int** id = (**int**)dgv\_buildings.CurrentRow.Cells["IdBuilding"].Value;

**using** (**var** db = **new** mriContext())

{

**var** buil\_to\_del = db.Buildings.Where(a => a.IdBuilding == id).First();

db.Remove(buil\_to\_del);

dgv\_buildings.DataSource = **null**;

db.SaveChanges();

}

}

**using** (**var** db = **new** mriContext())

{

**var** new\_build = **new** Building()

{

IdBuilding = bID,

BuildingAddress = dgv\_buildings.Rows[e.RowIndex].Cells["BuildingAddress"].Value.ToString(),

BuildingType = dgv\_buildings.Rows[e.RowIndex].Cells["BuildingType"].Value.ToString(),

BuildingFloors = **int**.Parse(dgv\_buildings.Rows[e.RowIndex].Cells["BuildingFloors"].Value.ToString())

};

db.Update(new\_build);

db.SaveChanges();

}

При редактировании, если происходит ошибка несоответствия типов данных, то dataGridView вызывает событие DataError и уже там происходит обработка данных ошибок:

**private** **void** **dgv\_buildings\_DataError**(**object** sender, DataGridViewDataErrorEventArgs e)

{

dgv\_clinics.Enabled = **false**;

dgv\_buildings.Rows[e.RowIndex].ErrorText = $"Значение в столбце Количество этажей должно быть целочисленным";

}

Как только происходит выделение какой-либо строки в таблице зданий, отображаются данные о клиниках, в этом здании, запрос выглядит следующим образом:

**var** result = (**from** Clinic **in** db.Clinics

**where** Clinic.IdBuilding == bui\_id

**select** Clinic).ToList();

Для связи с зданием, при добавлении новой клиники в соответствующую форму передается id выбранного здания.

Работа с сотрудниками и должностями выглядит аналогичным образом, что и с зданиями. Отличие есть в работе с должностями, т.к. в БД поле названия должности является первичным ключом, то его редактирование запрещено.

Стоит рассмотреть переход на форму с трудоустройством. На ней отображаются работники, работающие в данной клинике и их должности. Соединение происходит с помощью таблицы «emp\_pos». Так же выбор данных происходит через анонимный тип:

**using** (**var** db = **new** mriContext())

{

**var** result = (**from** Employer **in** db.Employers

**join** EmpPo **in** db.EmpPos on Employer.IdEmployer **equals** EmpPo.IdEmployer

**where** EmpPo.IdClinic==clinic

**select** **new** { Employer.IdEmployer,

Employer.EmployerName,

Employer.EmployerSurname,

Employer.EmployerMiddleName,

Employer.EmployerSex,

Employer.EmployerBirthday,

Employer.EmployerExperience,

EmpPo.Position}).ToList();

dgv\_work.DataSource = result;

}

Редактирование связей происходит в отдельной форме, в которой можно выбрать клинику и должность, но не сотрудника. Это довольно логично, ведь, выбирая сотрудника на форме трудоустройства и изменяя данные, подразумевается, что изменяются данные для этого сотрудника. При добавлении таких ограничений нет, добавить можно в любую клинику, любого сотрудника на любую должность, при условии, что данный сотрудник еще не работает в данной клинике. Форма выглядит следующим образом:

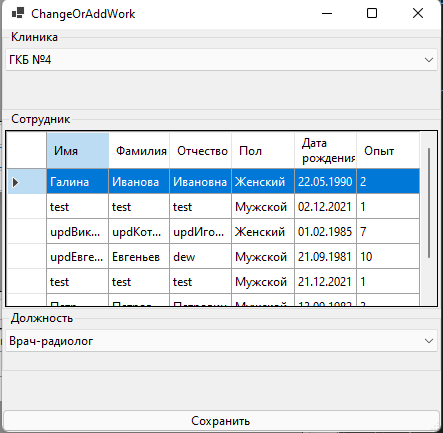


Рис 6 – Форма изменения данных о трудоустройстве

Так же при изменении действует немного иная логика. Все поля таблицы emp\_pos являются частью первичного ключа, что запрещает редактирование. По этой причине все части первичного ключа редактируемой записи передаются в данную форму. Создается копия редактируемого объекта и объект с новыми данными. После проверок на ограничения, удаляется старая запись из базы данных и добавляется измененная.

**using** (**var** db = **new** mriContext())

{

**var** empToDel = db.EmpPos.Where(a => a.IdClinic == clin && a.IdEmployer == emp && a.Position == pos).FirstOrDefault();

**if**(empToDel==**null**)

{

MessageBox.Show("Данный работник еще не трудоустроен");

**return**;

}

EmpPo empPo = **new** EmpPo

{

IdClinic = clinic[cb\_clinic.SelectedIndex],

IdEmployer = **int**.Parse(dgv\_emp.CurrentRow.Cells["IdEmployer"].Value.ToString()),

Position = cb\_pos.Text

};

**var** isInDataBase = db.EmpPos.Where(a => a.IdClinic == empPo.IdClinic && a.IdEmployer == empPo.IdEmployer && a.Position == empPo.Position).FirstOrDefault();

**if** (isInDataBase != **null**)

{

MessageBox.Show("Этот сотрудник уже работает на данной должности в данной клинике!");

**return**;

}

db.Remove(empToDel);

db.SaveChanges();

db.Add(empPo);

db.SaveChanges();

}

Таблица пользователей так же имеет одну особенность при добавлении пользователя. Открывается форма авторизации, на которой становится видна и доступна возможность выбора роли, заблокированы все кнопки, кроме «Регистрация».

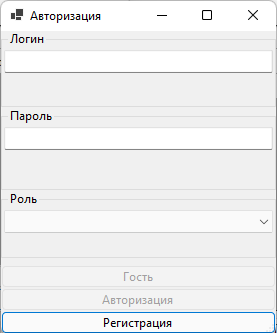


Рис 7 – Форма авторизации при добавлении пользователя

Стандартные запросы поиска содержат запросы: вывод всех врачей-радиологов; все государственные клиники. Какие-либо изменения в форме с данными, полученными этими запросами, запрещены.

# Тестирование корректности работы программы

1. Регистрация нового пользователя
2. Вход под ролью администратора, проверка, что все кнопки и функции доступны

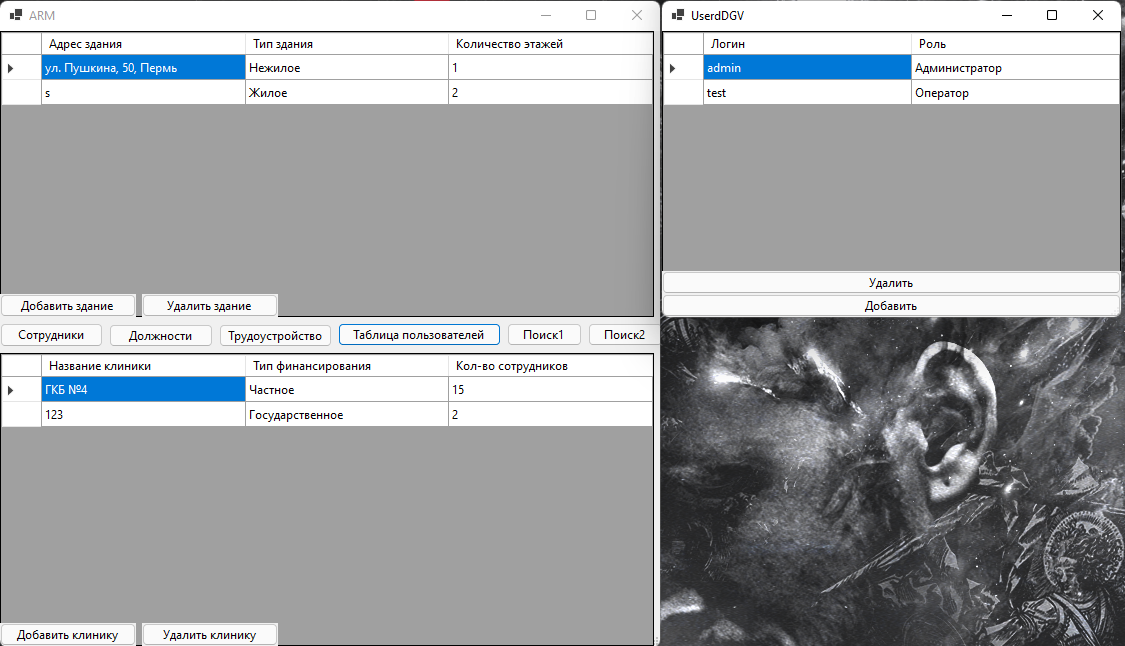


Рис 8 – Тест 2

1. Вход под ролью оператора, проверка, что все функции доступны, кроме таблицы пользователей

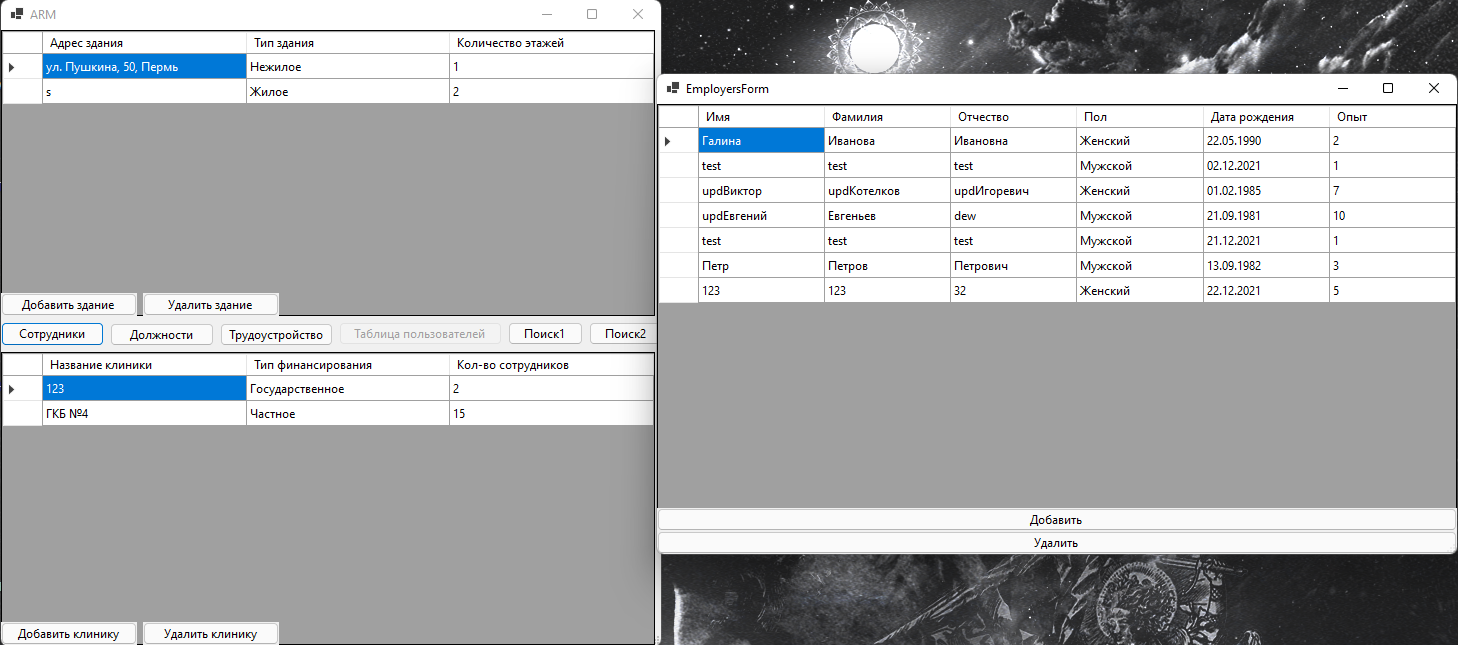
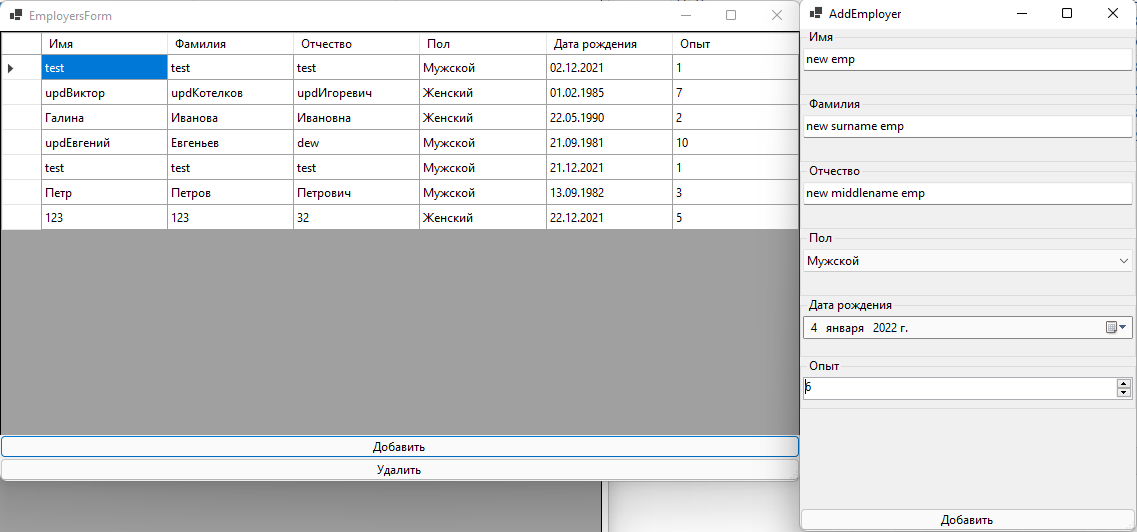


Рис 9 – Тест 3

1. Вход в гостевом режиме, проверка, что доступен просмотр, но запрещено редактирование
2. Изменение данных в таблице зданий
3. Добавление нового здания
4. Удаление здания
5. Изменение данных в таблице клиник
6. Добавление новой клиники
7. Удаление клиники
8. Изменение данных в таблице сотрудников
9. Добавление нового сотрудника



**Рис 10 – Тест 12**

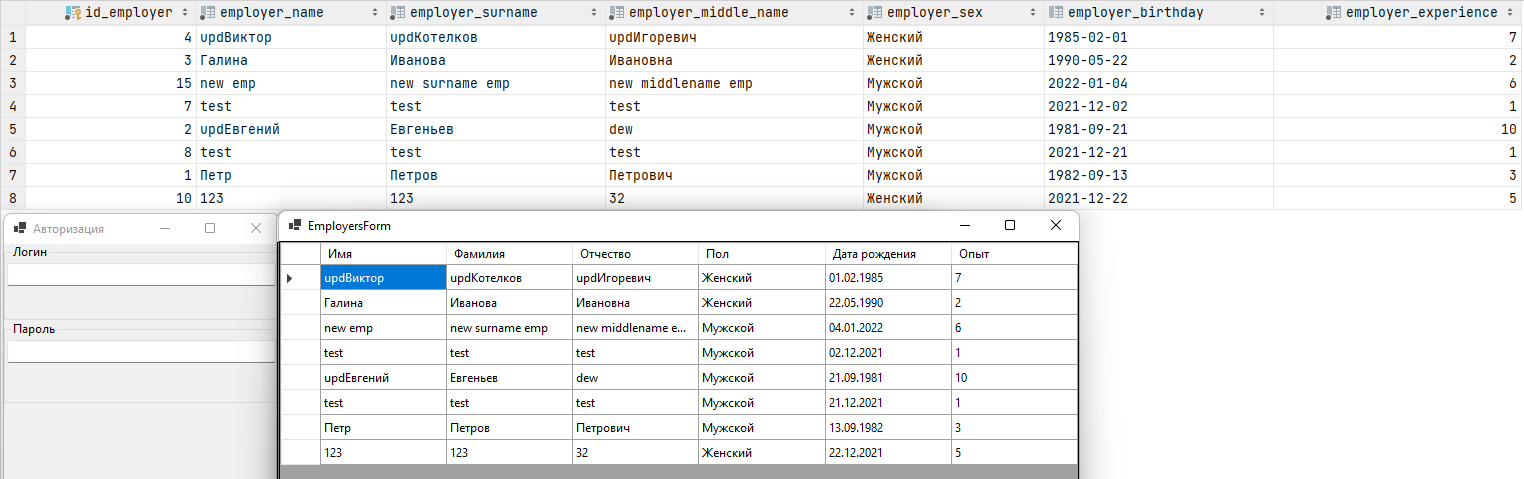


Рис 11 – Результат теста 12

1. Удаление сотрудника
2. Изменение данных в таблице должностей
3. Добавление новой должности
4. Удаление должности

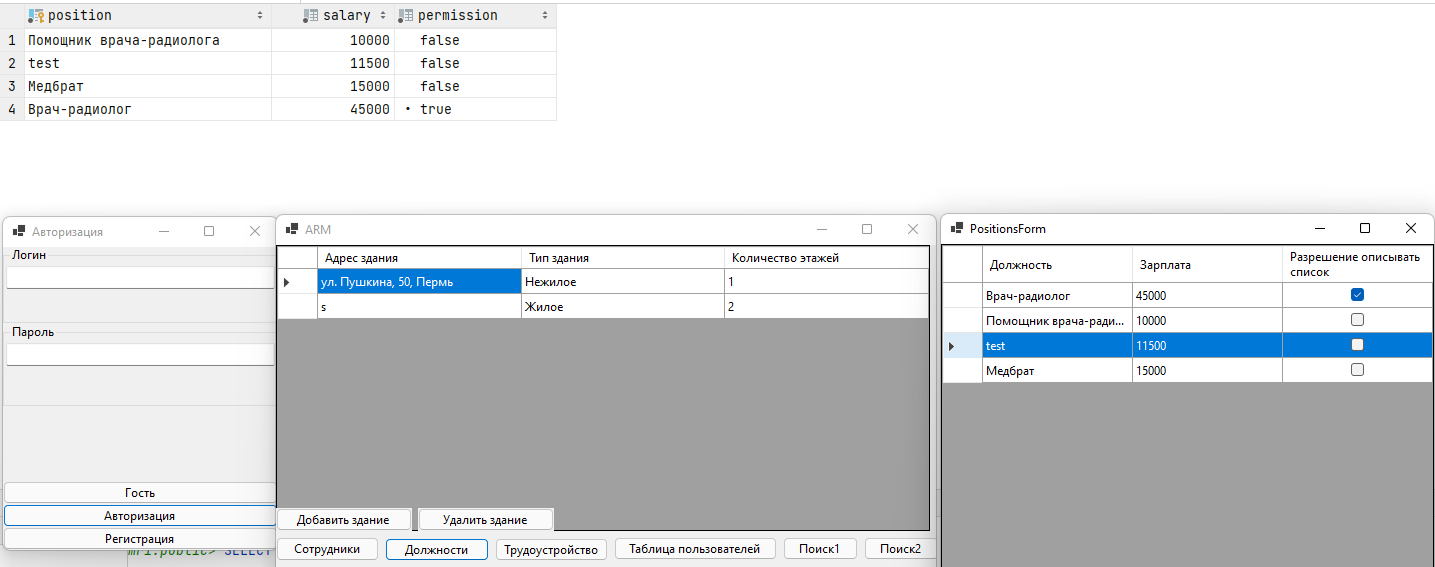


Рис 12 – Тест 16

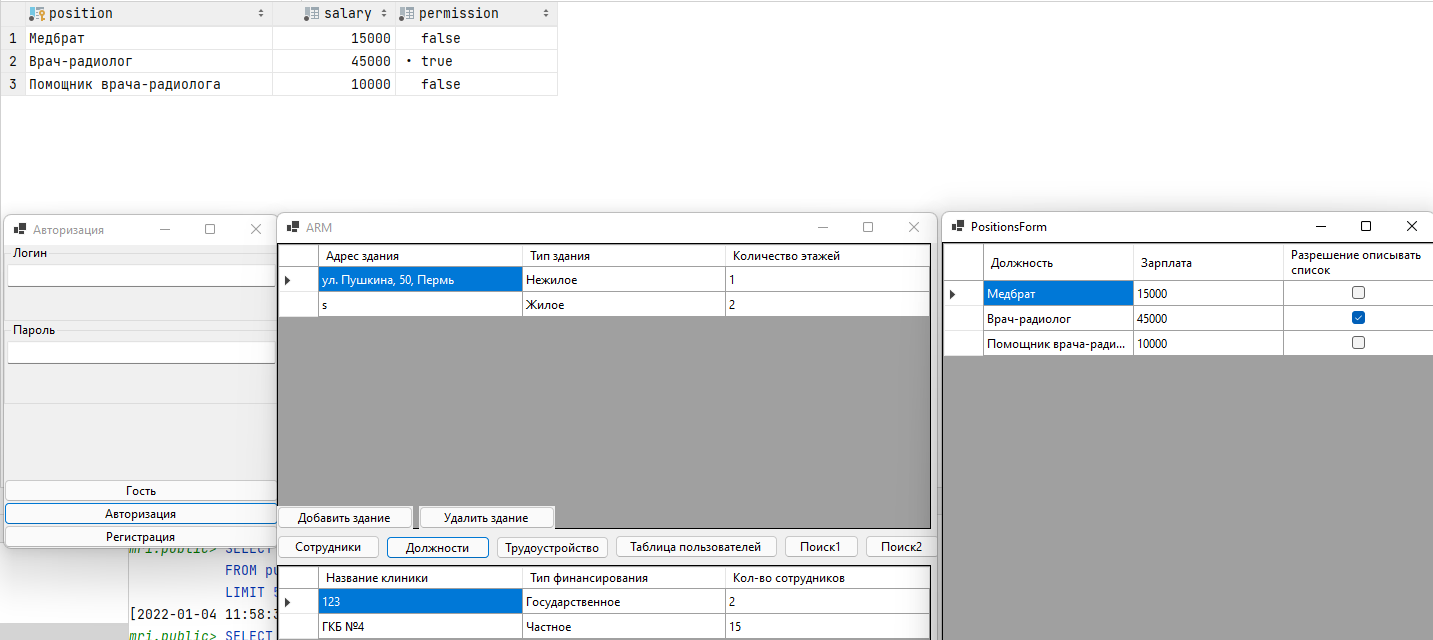


Рис 13 – Результат теста 16

1. Редактирование данных в таблице трудоустройство

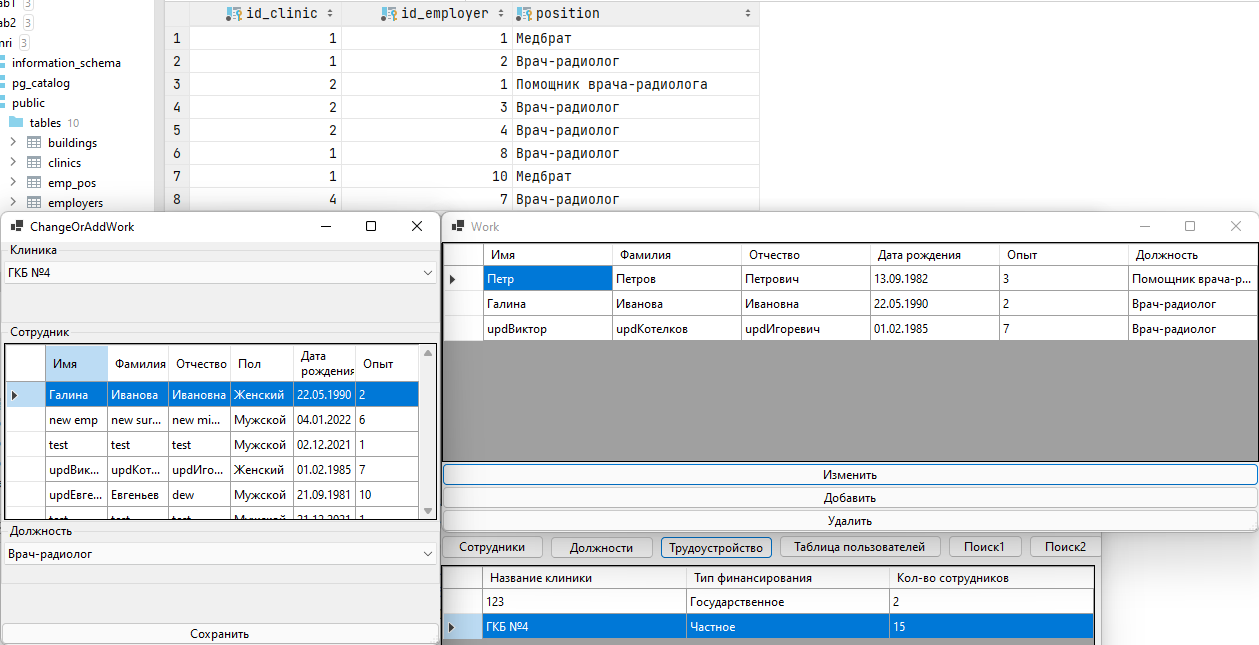


Рис 14 – Тест 17

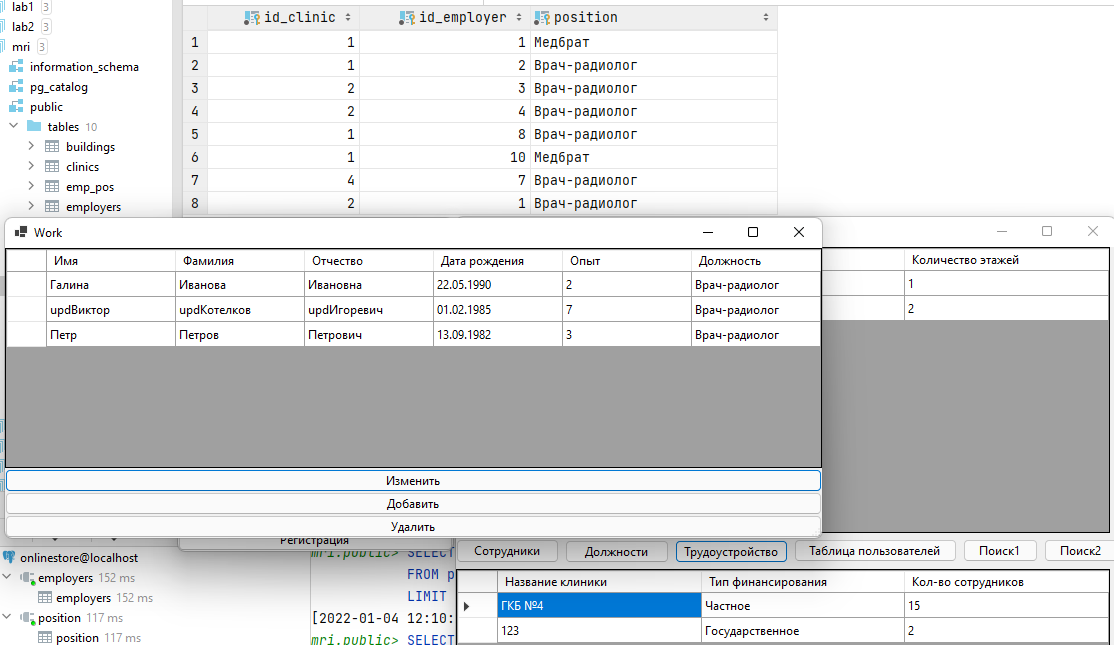


Рис 15 – Результат теста 17

1. Добавление в таблицу трудоустройство
2. Удаление из таблиц трудоустройства
3. Стандартный запрос поиска №1
4. Стандартный запрос поиска №2
5. Добавление в таблицу пользователей
6. Редактирование в таблице пользователей
7. Удаление из таблицы пользователей