МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»  
  
Факультет Информационных технологий   
Кафедра Информационных систем и технологий   
Специальность 1-40 05 01 Информационные системы и технологии   
Специализация 1-40 05 01 03 Информационные системы и технологии   
(издательско-полиграфический комплекс)   
  
  
**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**«Разработка базы данных «Помощник председателя  
товарищества собственников жилья»»

Выполнил студент Кашперко Василиса Сергеевна   
 (Ф.И.О.)

Руководитель проекта асс., Сазонова Д.В.   
 (учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В.   
 (учен. степень, звание, должность, Ф.И.О., подпись)

Консультант: асс., Сазонова Д.В.   
 (учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер: асс., Сазонова Д.В.   
 (учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Минск 2023

**Содержание**

[Введение 5](#_Toc134766270)

[1 Постановка задачи 7](#_Toc134766271)

[1.1 Аналитический обзор аналогов и литературных источников 7](#_Toc134766272)

[1.2 Анализ и разработка функциональных требований 10](#_Toc134766273)

[2 Разработка модели базы данных 12](#_Toc134766274)

[3 Разработка необходимых объектов 14](#_Toc134766275)

[3.1 Таблицы 14](#_Toc134766276)

[3.2 Пользователи 16](#_Toc134766277)

[3.3 Хранимые процедуры 17](#_Toc134766278)

[3.4 Индексы 17](#_Toc134766279)

[4 Описание процедур экспорта и импорта 18](#_Toc134766280)

[5 Тестирование производительности 19](#_Toc134766281)

[6 Описание технологии резервного копирования и восстановления и ее применения в базе данных 21](#_Toc134766282)

[7 Руководство пользователя 24](#_Toc134766283)

[Заключение 32](#_Toc134766284)

[Cписок используемых источников 33](#_Toc134766285)

[ПРИЛОЖЕНИЕ А 34](#_Toc134766286)

[ПРИЛОЖЕНИЕ Б 37](#_Toc134766287)

[ПРИЛОЖЕНИЕ В 39](#_Toc134766288)

[ПРИЛОЖЕНИЕ Г 62](#_Toc134766289)

[ПРИЛОЖЕНИЕ Д 63](#_Toc134766290)

[ПРИЛОЖЕНИЕ Е 65](#_Toc134766291)

# ВВЕДЕНИЕ

Информационные системы и технологии все увереннее становятся неотъемлемой частью всех сфер жизнедеятельности современного человека. Все потому, что именно они дают возможность современному человеку жить в комфорте и получать мгновенный доступ к нужной информации, не теряясь в стопках бумаг, блокнотах и других бумажных носителях информации.

Также информационные системы позволяют структурировать данные, в кратчайшие сроки отобрать искомую информацию наиболее удобными способами, избежать ошибок при заполнении данных, обеспечивая целостность, и предотвратить потерю информации в бытовых ситуациях.

В потребности организации подобных систем нуждаются не только государственные структуры, большие влиятельные компании и обширные сети, но и юридические, физические и должностные лица, то есть обычные люди со своей спецификой повседневных и профессиональных задач. Каждая организация или даже просто человек нуждаются в быстром доступе к информации, которая в настоящее время является ценным ресурсом. Чтобы обеспечить своевременный доступ к информации, организации часто используют базы данных в качестве распорядителей информации.

Базы данных хранят информацию в структурированном виде и обеспечивают ее надежное хранение и доступность. Почти каждая современная организация нуждается в базе данных, которая отвечает требованиям по хранению, управлению и администрированию данных.

Не исключением является и человек, занимающий должность председателя товарищества собственников жилья (далее – председатель ТСЖ). Председатель ТСЖ – это выборное должностное лицо, которое обеспечивает выполнение решений правления ТСЖ. Главная задача председателя – защита интересов собственников квартир в любых инстанциях. Между председателем и товариществом заключается трудовой договор, где подробно описываются закрепленные за ним обязанности, например:

* знать действующее законодательство страны (нормативно-правовые акты, относящиеся к деятельности сообщества, устав ТСЖ и должностную инструкцию);
* контролировать своевременную оплату жильцами коммунальных услуг и членских взносов;
* реализовывать решения, принятые на общем собрании;
* контролировать поддержание порядка в жилом доме;
* управлять персоналом по хозяйственной работе;
* работать с документацией и быть осведомленным о всех собственниках квартир;
* предоставлять отчет о проделанной работе, тратах на поддержание общедомового имущества.

В последние годы структуры данных стали более сложными. В дополнение к простым числам и текстовым строкам, данные теперь могут содержать мультимедийные документы, графические образы, хронологические ряды, процедурные или активные данные, а также множество других сложных форм информации.

Существует множество технологий доступа к данным и серверам баз данных, каждая из которых имеет свои особенности. Современные приложения обработки данных предназначены для работы с большим количеством пользователей и позволяют им работать удаленно от сервера базы данных.

Темой данного курсового проекта является разработка базы данных «Помощник председателя товарищества собственников жилья» с реализацией технологии резервного копирования и восстановления.

База данных «Помощник председателя товарищества собственников жилья» будет содержать соответствующие таблицы, процедуры, функции и предназначена для взаимодействия с источником данных. Взаимодействие подразумевает получение данных, их представление для просмотра пользователем, редактирование в соответствии с реализованными в программе алгоритмами и возврат обработанных данных обратно в базу данных.

База данных «Помощник председателя товарищества собственников жилья», удовлетворяющего профессиональные потребности председателя ТСЖ. Также предоставляет возможность администраторам своевременно вносить изменения.

В соответствии с заданием курсового проекта для проектирования базы данных используется система управления базами данных Oracle Database 12c.

В качестве интерфейса прикладного программирования был выбран обширный API-интерфейс – Windows Presentation Foundation (WPF), предназначенный для создания настольных программ с графически насыщенным пользовательским интерфейсом.

Для работы с WPF использовался объектно-ориентированный язык программирования с С-подобным синтаксисом – С#, разработанный для создания приложений на платформе Microsoft .NET Framework.

Содержание данной пояснительной записки отражает этапы выполнения курсового проекта.

1 Постановка задачи

Задача проекта: разработать архитектуру базы данных, создать процедуры и функции, взаимодействие с которыми будет понятно любому пользователю.

В соответствии с заданием курсового проекта следует не только создать базу данных, но и разработать программное средство, которое должно в полной мере демонстрировать возможности базы данных.

Для того, чтобы сформировать окончательные требования к проектируемому программному средству сначала рассмотрим прототипы из той же области.

1.1 Аналитический обзор аналогов и литературных источников

Немаловажным этапом в разработке программного продукта является аналитический обзор прототипов и литературных источников.

На сегодняшний день аналогичных и доступных в сети программных средств под операционные системы, подходящие для компьютеров или ноутбуков, существует не большое количество.

Мною были рассмотрены несколько программных средств:

«Председатель ТСЖ» – программа предназначена для использования в товариществах собственников жилья (ТСЖ), которые ведут как расчет квартплаты и прочих коммунальных услуг, так и паспортный учет. Программа написана на встроенном языке программирования 1С.

Интерфейс «Председатель ТСЖ» представлен на рисунке 1.1.

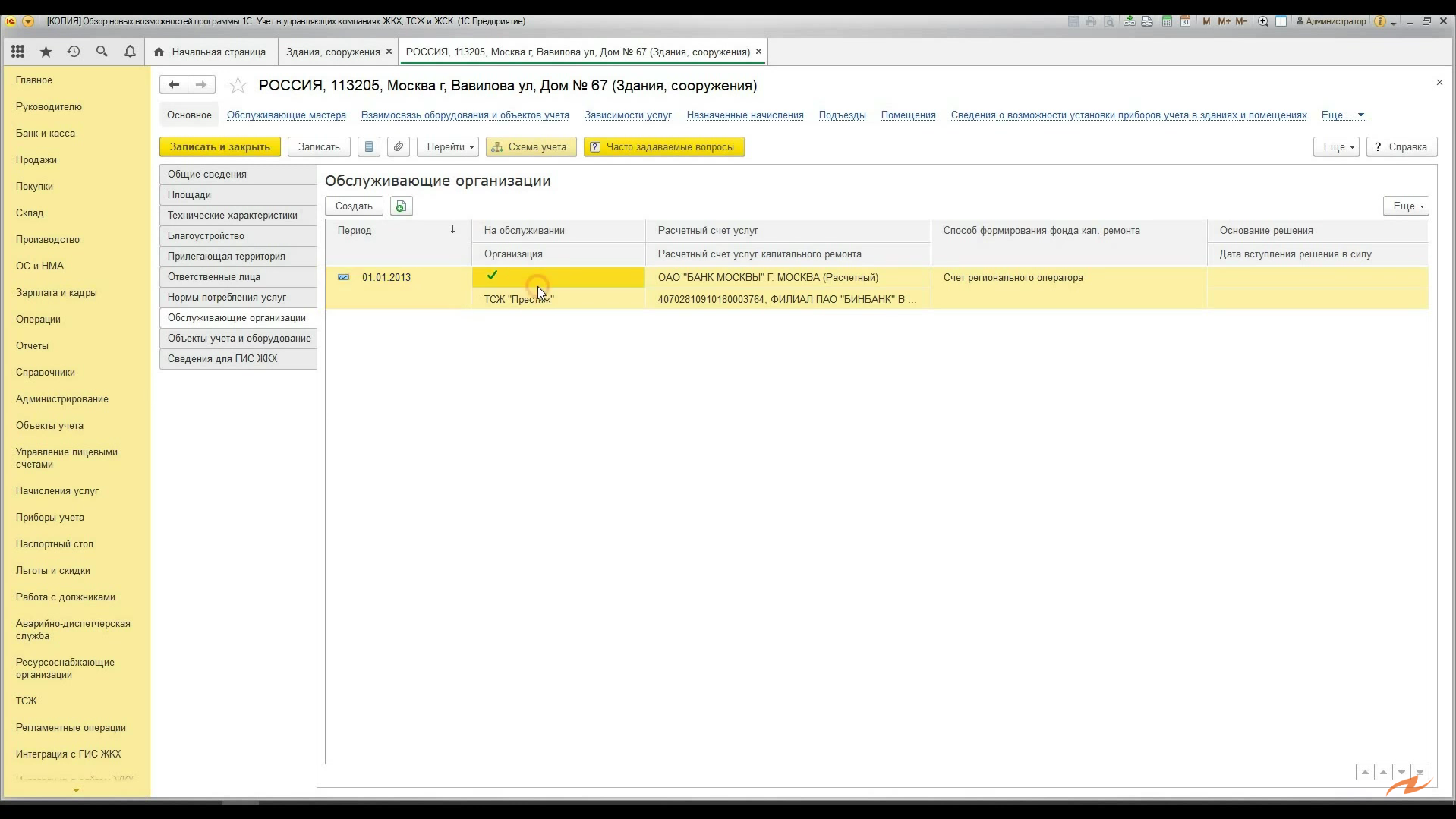


Рисунок 1.1 – Интерфейс программного средства «Председатель ТСЖ»

Проанализировав «Председатель ТСЖ», можно выделить основные минусы и плюсы данного программного средства.

Основные минусы:

* сложный интерфейс;
* стоимость.

Основные плюсы:

* широкое использование не только среди председателей товарищества собственников жилья, но и ЖКХ и ЖСК;
* возможность установить мобильное приложение и синхронизировать данные.

«Председатель 365» - мобильное приложение, разработано для облегчения работы председателя ТСЖ. Воспользоваться приложением не удалось, так как после авторизации, предлагают только вариант платной подписки. Поэтому, изучить приложение мне помогли скриншоты и видео из видео-хостингов.

Интерфейс одной из доступных страниц приложения «Председатель 365» представлен на рисунке 1.2.

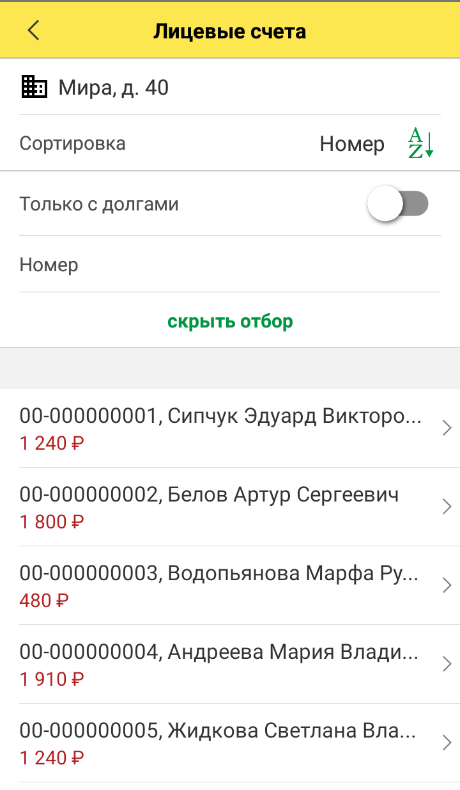


Рисунок 1.2 − Интерфейс «Председатель 365»

На рисунке 1.3 представлен скриншот окна со всеми адресами, введенными пользователем, приложения «Председатель 365».

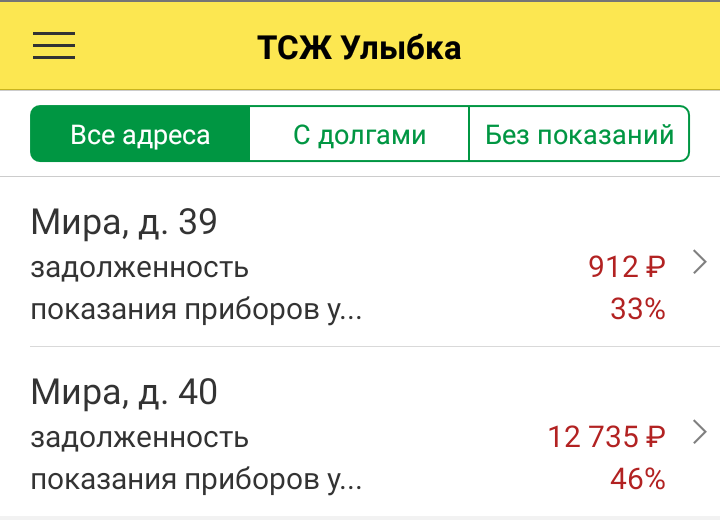


Рисунок 1.3 − Интерфейс «Председатель 365»

Проанализировав «Председатель 365», можно выделить основные минусы и плюсы данного программного средства.

Основные минусы:

* загруженный интерфейс;
* негативные отзывы покупателей подписки приложения;
* стоимость.

Основные плюсы:

* достаточно хороший UI;
* возможность загружать свои данные в облако.

«Домá» - мобильное приложение, разработано для облегчения работы председателя ТСЖ, помогающее своевременно получать заявки от жильцов, оплату жильцами членского взноса.

Интерфейс нескольких из доступных для изучения страниц приложения «Домá» представлен на рисунке 1.4.

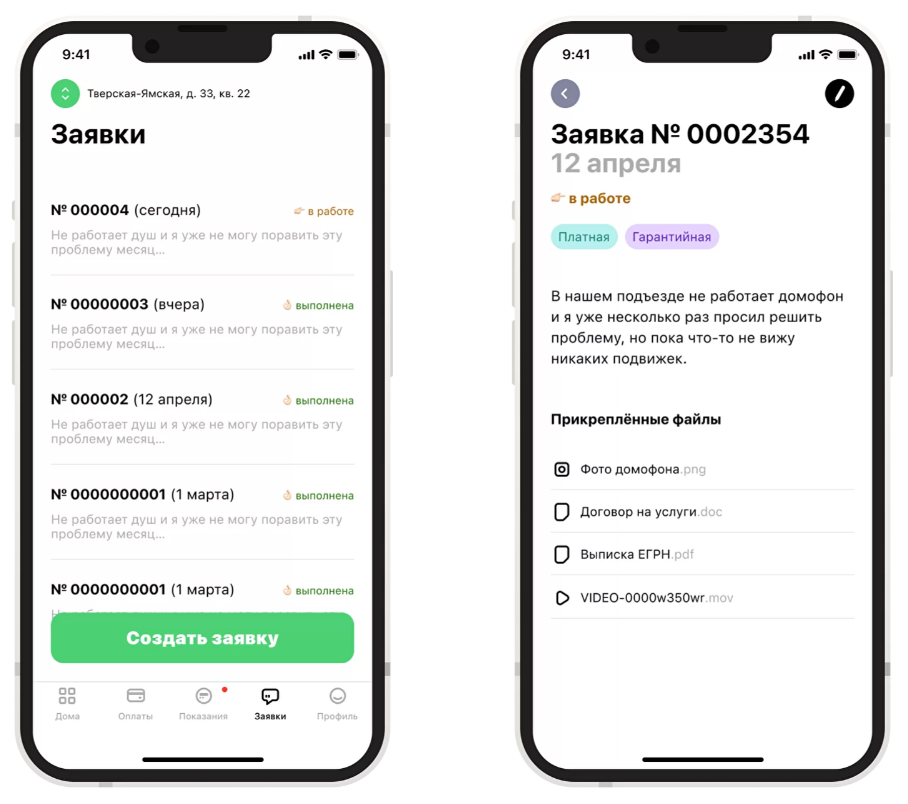


Рисунок 1.4 − Интерфейс «Домá»

Проанализировав «Председатель 365», можно выделить основные минусы и плюсы данного программного средства.

Основные минусы:

* стоимость;
* в приложении нет функций, сокращающих бумажную работу председателя.

Основные плюсы:

* хороший UI;
* возможность загружать свои данные в облако;
* возможность получать заявки от жильцов;
* возможность контролировать оплату членских взносов.

Таким образом был выполнен анализ необходимого функционала, предоставленный аналогами разрабатываемого приложения.

1.2 Анализ и разработка функциональных требований

Анализ требований – процесс сбора требований к программному обеспечению, их систематизации, документирования, анализа, выявления противоречий, неполноты, разрешения конфликтов в процессе разработки программного обеспечения.

Цель анализа требований в проектах – получить максимум информации о заказчике и специфике его задач, уточнить рамки проекта, оценить возможные риски.

На этом этапе происходит идентификация принципиальных требований методологического и технологического характера, формулируются цели и задачи проекта, а также определяются критические факторы успеха, которые впоследствии будут использоваться для оценки результатов внедрения.

Определение и описание требований – шаги, которые во многом определяют успех всего проекта, поскольку именно они влияют на все остальные этапы.

Различают три уровня требований к проекту:

* бизнес-требования;
* пользовательские требования;
* функциональные требования.

Бизнес-требования содержат высокоуровневые цели организации или заказчиков системы. Как правило, их высказывают те, кто финансируют проект, покупатели системы, менеджер реальных пользователей, отдел маркетинга.

Курсовой проект не подразумевает наличие заказчика, который мог бы выдвинуть бизнес-требования, поэтому в качестве таких высокоуровневых требований можно рассматривать общие требования к разрабатываемому средству.

К их числу относятся:

* простота и лёгкость интерфейса;
* использование принципов объектно-ориентированного программирования;
* использование архитектурных шаблонов проектирования;
* использование системы управления базами данных (СУБД);

Весь дальнейший процесс проектирования и разработки программного средства должен находиться в очерченных бизнес-требованиями границах.

Следующими требованиями являются требования пользователей приложения, а именно председателя ТСЖ и бухгалтера, закрепленного за председателем.

Данные требования описывают цели и задачи, которые пользователям позволит решить система. Таким образом, в пользовательских требованиях указано, что клиенты смогут делать с помощью системы.

Пользователь данного программного решения должен иметь возможности, соответствующие его роли.

Возможности председателя ТСЖ:

* + выполнять регистрацию и авторизацию;
  + поддерживать работу c базой данных;
  + добавлять и изменять информацию о жилых домах и собственниках квартир;
  + просматривать информацию о жилых домах и собственниках квартир;
  + выполнять поисковые запросы;
  + добавлять контакты работников дома и организаций в записную книжку;
  + выполнять регистрацию бухгалтера для определенного дома.

Возможности бухгалтера ТСЖ:

* + выполнять авторизацию;
  + просматривать информацию о собственниках квартир;
  + добавлять и обновлять сведения о текущих задолженностях собственников.

После проведения анализа были выявлены следующие функциональные требования программного средства:

* архитектура приложения должна соответствовать шаблонам проектирования, таким как: MVVM, Command;
* вся информация должна храниться в базе данных;
* приложение должно производить валидацию вводимых пользователем данных;
* приложение должно корректным образом обрабатывать возникающие исключительные ситуации: отображать понятное для пользователя сообщение о возникшей ошибке;
* приложение должно предоставлять пользователям возможность создания нового аккаунта в виде регистрационной формы;
* приложение должно предоставлять возможность пользователям проходить аутентификацию и входить в систему под соответствующим введенным данным пользовательским именем;
* приложение должно предоставлять возможность поиска жильцов и квартир с соответствующими данными по следующим критериям, как: фамилия, номер квартиры.

Таким образом, был проведен тщательный анализ требований к программному средству, который позволил разработать список функциональных требований. Разработка данной программной системы должна проводиться в соответствии с сформированными списком.

2 Разработка модели базы данных

Проектирование баз данных представляет собой процесс создания структуры базы данных и определения необходимых ограничений целостности.

Основные задачи проектирования базы данных:

* обеспечение хранения в БД всей необходимой информации;
* обеспечение возможности получения данных по всем необходимым запросам;
* сокращение избыточности и дублирования данных;
* обеспечение целостности базы данных.

Проектирование базы данных проходит через два основных этапа: концептуальное и логическое проектирование.

Концептуальное проектирование представляет собой создание семантической модели предметной области, то есть информационной модели на высоком уровне абстракции. В результате этого этапа создается ER-модель, которая не зависит от конкретной СУБД и модели данных. Основными понятиями ER-модели являются: сущность, связь и атрибут.

Сущность – реальный или представляемый объект, информация о котором должна сохраняться и быть доступной.

Связь – графически представляемая ассоциация между двумя сущностями. Связь может существовать между разными сущностями или быть рекурсивной (сущность связана с самой собой).

Атрибут сущности – любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности.

В рамках этого этапа была создана ER-модель, которая включает 8 сущностей:

* пользователь;
* дом;
* адрес;
* подъезд;
* квартира;
* жилец-собственник;
* номер телефона;
* контакт.

Также в ER-модели были определены необходимые связи. Например, между сущностями «пользователь» и «дом» была установлена связь один-ко-многим. Для каждой сущности были выделены атрибуты.

Например, для дома в качестве атрибутов были выделены такие характеристики, как «идентификатор дома», «идентификатор пользователя», «название дома», «количество квартир», «количество подъездов» и «идентификатор адреса».

Логическое проектирование – создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных.

Для реляционной модели данных логическая модель – набор схем отношений, обычно с указанием первичных ключей, а также «связей» между отношениями, представляющих собой внешние ключи.

На этапе логического проектирования учитывается специфика конкретной модели данных, но может не учитываться специфика конкретной СУБД.

Логическая модель базы со структурой связей представлена на рисунке 2.1.



Рисунок 2.1 – Логическая модель базы данных

Всего в базе данных содержится 8 таблиц.

В таблице Users хранятся все пользователи, зарегистрированные в приложении.

Таблица Houses содержит информацию о добавленных пользователем домах.

Таблица Addresses содержит информацию о адресах добавленных домов.

Таблица Porches хранит сгенерированные подъезды, чтобы упросить навигацию по квартирам в доме.

Таблица Flats содержит квартиры домов. Таблица Owners хранит информацию о собственниках квартир.

Таблица PhoneNumbers хранит телефонные номера как собственников, так и людей, добавленных в контакты.

Таблица Contacts содержит информацию о добавленных контактах.

3 Разработка необходимых объектов

3.1 Таблицы

Таблица – это совокупность связанных данных, хранящихся в структурированном виде в базе данных. Она состоит из столбцов и строк.

Столбцы таблицы называют полями; каждое поле характеризуется своим именем (названием соответствующего свойства) и типом данных, отражающих значения данного свойства. Каждое поле обладает определенным набором свойств (размер, формат и др.).

Поле базы данных – это столбец таблицы, включающий в себя значения определенного свойства.

В каждой таблице должно быть, по крайней мере, одно ключевое поле, содержимое которого уникально для любой записи в этой таблице.

Значения ключевого поля однозначно определяют каждую запись в таблице.

Для реализации базы данных «Помощник председателя товарищества собственников жилья» было разработано 8 таблиц: Users, Addresses, Houses, Porches, Flats, Owners, PhoneNumbers, Contacts. Они будут реализованы в СУБД Oracle DataBase 12c [1].

Таблица Users представляет список логинов и паролей всех пользователей и информации о них (таблица 3.1).

Таблица 3.1 – Столбцы таблицы Users

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| UserId | Идентификатор пользователя | number |
| Surname | Фамилия пользователя | nvarchar2 |
| Name | Имя пользователя | nvarchar2 |
| Patronymic | Отчество пользователя | nvarchar2 |
| Login | Логин | nvarchar2 |
| Password | Пароль | nvarchar2 |
| Role | Роль | number |
| AccountantId | Идентификатор бухгалтера | number |

Таблица Addresses представляет список адресов домов, добавленных пользователями (таблица 3.2).

Таблица 3.2 – Столбцы таблицы Addresses

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| AddressesId | Идентификатор адреса | number |
| Country | Страна | nvarchar2 |
| City | Город | nvarchar2 |
| District | Район | nvarchar2 |
| Street | Улица | nvarchar2 |
| HouseNumber | Номер дома | number |
| HousingNumber | Номер корпуса | nvarchar2 |

Таблица Houses представляет список домов, добавленных пользователями, находящихся по определенному адресу (таблица 3.3).

Таблица 3.3 – Столбцы таблицы Houses

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| HouseId | Идентификатор дома | number |
| HouseName | Имя дома, заданное пользователем | nvarchar2 |
| NumberOfFlats | Количество квартир | number |
| NumberOfPorches | Количество подъездов | number |
| AddressId | Идентификатор адреса | number |
| UserId | Идентификатор пользователя | number |

Таблица Porches представляет список подъездов, находящихся в доме. Столбцы данной таблицы описаны в таблице 3.4.

Таблица 3.4 – Столбцы таблицы Houses

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| PorchId | Идентификатор подъезда | number |
| PorchNumber | Номер подъезда | number |
| HouseId | Идентификатор дома | number |

Таблица Flats представляет список квартир, находящихся в подъездах дома, который добавил председатель (таблица 3.5).

Таблица 3.5 – Столбцы таблицы Flats

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| FlatId | Идентификатор квартиры | number |
| FlatNumber | Номер квартиры | number |
| PorchId | Идентификатор подъезда | number |

Таблица Owners представляет список собственников жильцов, относящихся к квартирам (таблица 3.6).

Таблица 3.6 – Столбцы таблицы Owners

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| OwnerId | Идентификатор собственника | number |
| Surname | Фамилия собственника | nvarchar2 |
| Name | Имя собственника | nvarchar2 |
| Patronymic | Отчество собственника | nvarchar2 |
| AdditionalInfo | Дополнительная информация о собственнике | nvarchar2 |
| CurentDebt | Текущая задолженность | number |
| PhoneNumber | Номер телефона | number |
| OwnerStatusId | Идентификатор статуса жильца собственника | number |
| FlatId | Идентификатор квартиры | number |

Таблица PhoneNumbers представляет список номеров телефона как жильцов собственников, так и добавленных пользователем контактов (таблица 3.7).

Таблица 3.7 – Столбцы таблицы PhoneNumbers

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| PhoneNumberId | Идентификатор номеров телефона | number |
| MobilePhone | Мобильный номер телефона | nvarchar2 |
| HomeNumber | Домашний номер телефона | nvarchar2 |
| AdditionalPhone | Дополнительный номер телефона | nvarchar2 |

Таблица Contacts представляет список контактов в записной книжке пользователя (таблица 3.8).

Таблица 3.8 – Столбцы таблицы Contacts

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ContactId | Идентификатор контакта | number |
| Surname | Фамилия контакта | nvarchar2 |
| Name | Имя контакта | nvarchar2 |
| Patronymic | Отчество контакта | nvarchar2 |
| Position | Должность | nvarchar2 |
| PhomeNumberId | Идентификатор номеров телефона | number |
| UserId | Идентификатор пользователя | number |

Скрипты создания таблиц и наложение ограничений целостности на них представлены в приложении А данной записки.

3.2 Пользователи

Пользователь базы данных – это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации.

При проектировании базы данных было использовано 3 пользователя: администратор, председатель и бухгалтер. Каждый пользователь имеет разные привилегии в соответствии с его позицией.

Администратор управляет общей информацией и данными базами данных, в том числе логинами и паролями пользователей. Однако, доступ со стороны приложения предоставляется только пользователям уровня председатель и бухгалтер. Администрирование базой данных осуществляется только на уровне самой базы данных.

Председатель добавляет дома, указывает информацию о собственниках и управляет своими записями. Бухгалтер закрепляется за домами, добавленными определенным председателем, он может добавлять и обновлять задолженность жильцов.

Скрипты создания пользователей и их ролей представлены в приложении Б данной записки.

3.3 Хранимые процедуры

Хранимая процедура – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере.

Таким образом, процедура будет принимать соответствующие аргументы при каждом ее вызове.

Все хранимые процедуры, созданные в данном курсовом проекте, содержат обработку исключений для того, чтобы ограничить пользователя от возможных ошибок.

При разработке курсового проекта было создано множество процедур для следующих целей:

* добавление/изменение/удаление пользователей;
* добавление/изменение/удаление домов и информации о них;
* добавление/изменение/удаление адресов и информации о них;
* добавление/изменение/удаление информации о квартирах;
* добавление/изменение/удаление информации о жильцах;
* добавление/изменение/удаление контактов в записной книге;
* поиск пользователей по фамилии;
* поиск пользователей по идентификатору;
* добавление квартир, распределенных по подъездам, в зависимости от количества квартир и подъездов в доме;
* экспорт и импорт таблицы в формат xml.

Все хранимые процедуры представлены в приложении В данной пояснительной записки.

Одна из самых главных процедур для данного курсового проекта расположена на странице 48 данной пояснительной записки – добавление подъездов и квартир по id дома. Процедура извлекает информацию о доме по номеру идентификатора, такую как: количество подъездов и квартир – несмотря на то, что подъезды и квартиры можно добавить и по отдельности, такая процедура позволяет автоматизировать процесс добавления подъездов и квартир, правильно разделяя квартиры, присваивая им нужный идентификатор подъезда. Таким образом можно сразу вставить, к примеру, 120 квартир, распределенные по 4 подъездам, и пользователю базы данных не придется их создавать вручную.

3.4 Индексы

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

Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать много времени.

Индексы, созданные для полей таблиц в рамках курсового проекта, находится в приложении Г.

4 Описание процедур экспорта и импорта

Формат XML используется для обмена данными в удобной и распространенной форме. В Oracle часто возникает потребность в импорте и экспорте XML-файлов.

В рамках курсового проекта было выполнено экспортирование данных в формат XML и импортирование данных из XML-файлов.

В данном проекте данные процедуры экспорта и импорта применяются для работы с таблицей Houses.

Код процедуры экспорта в XML формат представлен в листинге 4.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE ExportHousesToXML  IS  v\_xml XMLType;  v\_clob CLOB;  v\_file UTL\_FILE.FILE\_TYPE;  BEGIN  -- Выборка данных из таблицы Houses в XMLType  SELECT XMLElement(  "Houses",  XMLAgg(  XMLElement(  "House",  XMLForest(  HouseId AS "HouseId",  HouseName AS "HouseName",  NumberOfFlats AS "NumberOfFlats",  NumberOfPorches AS "NumberOfPorches",  AddressId AS "AddressId",  UserId AS "UserId"))))  INTO v\_xml  FROM Houses;  -- Преобразование XMLType в CLOB  v\_clob := v\_xml.getClobVal();  -- Создание и запись XML в файл  v\_file := UTL\_FILE.FOPEN('EXPORT\_DIR', 'houses.xml', 'W');  UTL\_FILE.PUT\_RAW(v\_file, UTL\_RAW.CAST\_TO\_RAW(v\_clob));  UTL\_FILE.FCLOSE(v\_file);    DBMS\_OUTPUT.PUT\_LINE('Export completed successfully.');  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error occurred while exporting houses to XML.');  DBMS\_OUTPUT.PUT\_LINE(SQLERRM);  END; |

Листинг 4 – Процедура экспорта данных из таблицы Houses в XML-формат

Полный код процедур экспорта и импорта данных в формат XML представлен в приложении Д данной записки.

5 Тестирование производительности

Производительность базы данных является ключевым фактором эффективности управленческих и коммерческих приложений.

Если операции поиска или записи данных выполняются медленно, это негативно сказывается на работе приложения. Для выяснения причины плохой производительности необходимо провести количественные измерения и определить источник проблемы.

Проблемы, связанные с выявлением узких мест производительности баз данных, напрямую связаны с метриками, методами измерения производительности и используемыми технологиями.

Для организации процесса тестирования баз данных тестировщики должны обладать хорошими знаниями SQL и языка манипулирования данными (DML), а также иметь ясное представление о внутренней структуре базы данных. Это является наилучшим и надежным способом тестирования базы данных, особенно для приложений с низким и средним уровнем сложности.

Такой подход не только обеспечивает высокое качество тестирования, но также развивает навыки написания SQL-запросов.

Для проверки производительности базы данных необходимо заполнить ее большим объемом различных данных и измерить время выполнения отдельного запроса.

Время, затраченное на выполнение такого блока, отображено в Dbms Output.

На рисунке 5.1 изображен вывод полученной информации об общем времени выполнения и общем времени ЦПУ в секундах.

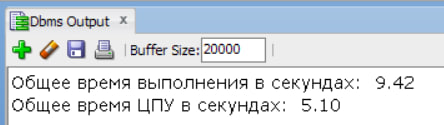


Рисунок 5.1 – Затраченное время на выполнение анонимного блока без индекса

Затем создадим индекс для таблицы Addresses на поле Country и изучим время выполнения аналогичного запроса.

На рисунке 5.3 показано затраченное время на выполнение такого же блока с индексом.

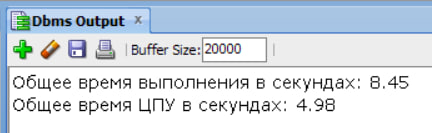


Рисунок 5.2 – Затраченное время на выполнение анонимного блока с индексом

Как можно заметить, общее время выполнения уменьшилось, но тем не менее даже без индекса данная база данных успешно прошла тест на производительность.

Для задачи тестирования производительности использовался анонимный блок и готовая процедура для вставки данных в таблицу, так как мы работаем с реальной базой данных.

Разработанный анонимный блок позволяет запустить процедуру InsertAddress 100000 раз за одно выполнение и затем откатить изменения, так как данный блок используется только для тестирования производительности (листинг 5.1).

|  |
| --- |
| set serveroutput on;  declare  v\_t1 number; -- время начала  v\_t2 number; -- время завершения  v\_cpu1 number; -- время ЦПУ до  v\_cpu2 number; -- время ЦПУ после  v\_t\_res number; -- общее время выполнения  v\_cpu\_res number; -- общее время выполнения CPU  begin  select t.hsecs  ,dbms\_utility.get\_cpu\_time  into v\_t1  ,v\_cpu1  from v$timer t;  -- запустим 100000 раз цикл, выполняющий процедуру вставки в таблицу Addresses  begin  for i IN 1..100000  loop  C##Vasilisa.InsertAddress('TEST', 'TEST', 'TEST', 'TEST', 123, 'A');  end loop;  end;  select t.hsecs  ,dbms\_utility.get\_cpu\_time  into v\_t2  ,v\_cpu2  from v$timer t;  v\_t\_res := v\_t2 - v\_t1;  v\_cpu\_res := v\_cpu2 - v\_cpu1;    dbms\_output.put\_line('Общее время выполнения в секундах: '||to\_char(v\_t\_res/100,'0.00'));  dbms\_output.put\_line('Общее время ЦПУ в секундах: '||to\_char(v\_cpu\_res/100,'0.00'));  rollback;  end; |

Листинг 5.1 – Анонимный блок для тестирования производительности

Таким образом было добавлено 100000 строк в таблицу Addresses.

Полный скрипт тестирования базы данных на производительность представлен в приложении Е данной записки.

6 Описание технологии резервного копирования и восстановления и ее применения в базе данных

В процессе разработки проекта была использована технология «Резервное копирование и восстановление данных».

Технология резервного копирования и восстановления (backup and recovery) в базе данных Oracle 12c предназначена для обеспечения сохранности и целостности данных при различных сбоях, таких как отказ оборудования, ошибки оператора или разрушения данных.

Резервное копирование – это процесс создания точной копии данных, которые находятся в базе данных, а восстановление – процесс восстановления базы данных из резервной копии.

В Oracle 12c существуют различные виды резервного копирования, такие как полное копирование, инкрементное копирование и дифференциальное копирование.

Полное копирование включает в себя все данные в базе данных, инкрементное копирование выполняет копирование только тех данных, которые изменились с момента последнего копирования, а дифференциальное копирование копирует только те данные, которые изменились с момента последнего полного копирования.

Одним из наиболее распространенных способов резервного копирования в Oracle 12c является использование инструмента управления резервным копированием (Recovery Manager, RMAN), который позволяет выполнять все этапы процесса резервного копирования и восстановления.

RMAN использует специальный формат файла копии данных (backup set), который содержит все необходимые данные для восстановления базы данных.

Для начала осуществляется подготовка к работе с RMAN: осуществляется подключение к базе данных.

Запуск RMAN и подключение к базе данных с помощью команды "CONNECT" с указанием учетных данных администратора базы данных или, например, пользователя, наделенного достаточными привилегиями (рисунок 6.1).

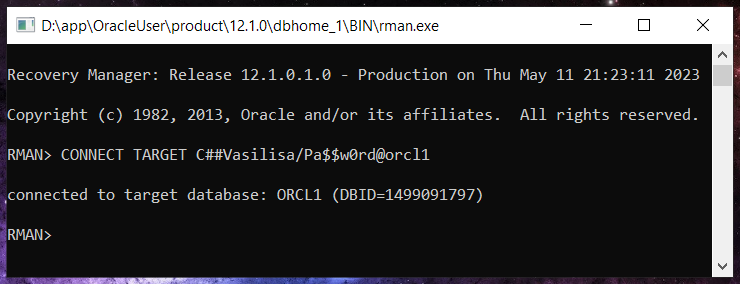


Рисунок 6.1 – Проверка подключения со стороны ранее созданного пользователя

RMAN предоставляет мощные средства для создания и управления резервными копиями базы данных.

В листинге 6.1 представлены последовательно введенные команды для подготовки к резервному копированию в RMAN.

|  |
| --- |
| RMAN> connect target /  \* отображение текущей политики сохранения для базы данных \*  RMAN> show retention policy;  \* настройка политики сохранения – RMAN будет хранить не менее двух копий каждого файла резервной копии или архивного журнала \*  RMAN> CONFIGURE RETENTION POLICY TO REDUNDANCY 2;  \* проверка состояния архивных журналов в базе данных \*  RMAN> crosscheck archivelog all; |

Листинг 6.1 – Последовательность команд для подготовки к резервному копированию

В листинге 6.2 представлены команды для подготовки базы данных к резервному копированию в SQL Plus.

|  |
| --- |
| sqlplus / as sysdba  SHUTDOWN IMMEDIATE;  STARTUP MOUNT;  ALTER DATABASE ARCHIVELOG;  ALTER DATABASE OPEN; |

Листинг 6.2 – Команды в SQL Plus

Потом определяются цели резервного копирования: указывается, какие компоненты базы данных вы хотите скопировать, например, данные, контрольные файлы, журналы перезаписи и другие файлы.

И наконец, выполняется резервное копирование: используются команды RMAN для запуска операции резервного копирования. В листинге 6.3 представлена команда, осуществляющая резервное копирование.

|  |
| --- |
| RMAN> run {backup as compressed backupset database spfile plus archivelog; delete noprompt obsolete; };  \* "Compressed backupset" указывает, что резервные копии будут сжаты для экономии места на диске или в хранилище \*  \* Ключевое слово "spfile" указывает RMAN на включение в резервное копирование файл настроек сервера базы данных (SPFILE) \*  \* "Plus archivelog" указывает RMAN на включение архивных журналов в резервное копирование \* \* "Noprompt" означает, что RMAN будет выполнять удаление без запроса подтверждения на удаление устаревших (obsolete) файлов резервной копии у пользователя \* |

Листинг 6.3 – Команда, осуществляющая резервное копирование

По завершению процесса резервного копирования созданные файлы можно найти по такому расположению: ...\App\OracleUser\Fast\_Recovery\_Area\orcl1\\*.

На рисунке 6.2 изображена структура файлов, расположенная по вышеупомянутому пути, созданная при помощи резервного копирования.

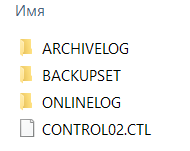


Рисунок 6.2 – Структура файлов, созданная резервным копированием

По завершении операций резервного копирования и управления копиями, можно отключиться от RMAN с помощью команды "DISCONNECT".

Для того, чтобы намеренно удалить файлы базы данных, нужно перейти по такому расположению: ..\App\OracleUser\oradata\orcl1\\* – и затем удалить их.

Для восстановления данных сначала так же выполняется подключение к базе данных: запуск RMAN и подключение к базе данных с помощью команды "CONNECT" с указанием учетных данных администратора базы данных.

Потом определяются цели восстановления: указывается, какие компоненты базы данных вы хотите восстановить. Например, можно восстановить полную базу данных, отдельные таблицы, контрольные файлы.

Наконец, происходит выполнение операции восстановления: использование команды RMAN для запуска операции восстановления. В листинге 6.4 представлены команды, осуществляющие восстановление файлов базы данных из резервных файлов.

|  |
| --- |
| create spfile from pfile-'D:\app\OracleUser\admin\orcl1\pfile\ init.ora.852022235434'  connect target /  restore controlfile from 'D:\app\OracleUser\fast\_recovery\_area\ orcl1\BACKUPSET\2023\_05\_12\O1\_MF\_ANNNN\_TAG20230504T083327\_L5VMX7 R2\_.BKP'  restore spfile to 'D:\app\OracleUser\fast\_recovery\_area\orcl1\ BACKUPSET\2023\_05\_12\O1\_MF\_ANNNN\_ TAG20230504T083327\_L5VMX7R2\_.BKP'  restore database from tag-'TAG20230504T083327';  restore database untill scn-1110855; |

Листинг 6.3 – Команда, осуществляющая резервное копирование

По завершении операции восстановления и управления, вы можете отключиться от RMAN с помощью команды "DISCONNECT".

Важно отметить, что резервное копирование и восстановление данных - сложный процесс, и для более точных и подробных инструкций, настройки и параметры необходимо знать все возможности инструмента RMAN.

7 Руководство пользователя

В результате работы над курсовым проектом была разработана полноценная база данных, состоящая из таблиц. Были разработаны процедуры и роли для пользователей различных привилегий. Также было разработано приложение, позволяющее работать с базой данных.

После запуска приложения в первую очередь открывается окно со страницей входа и предложением о регистрации.

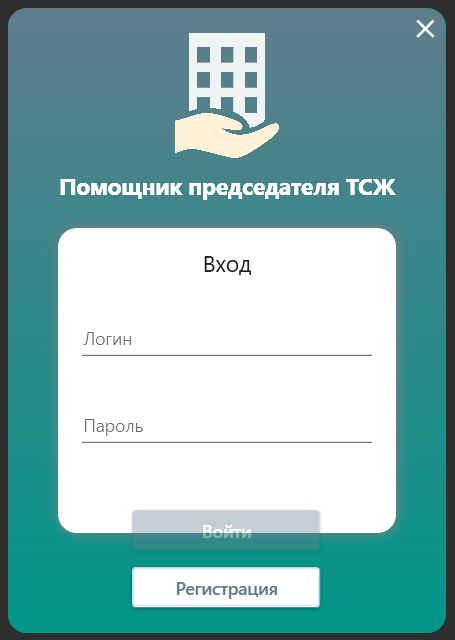


Рисунок 7.1 – Страница входа

После того, как пользователь вошел в аккаунт, открывается главная страница с уже добавленными домами или возможностью добавления нового дома, если пользователь только зарегистрировался (рисунок 7.2).

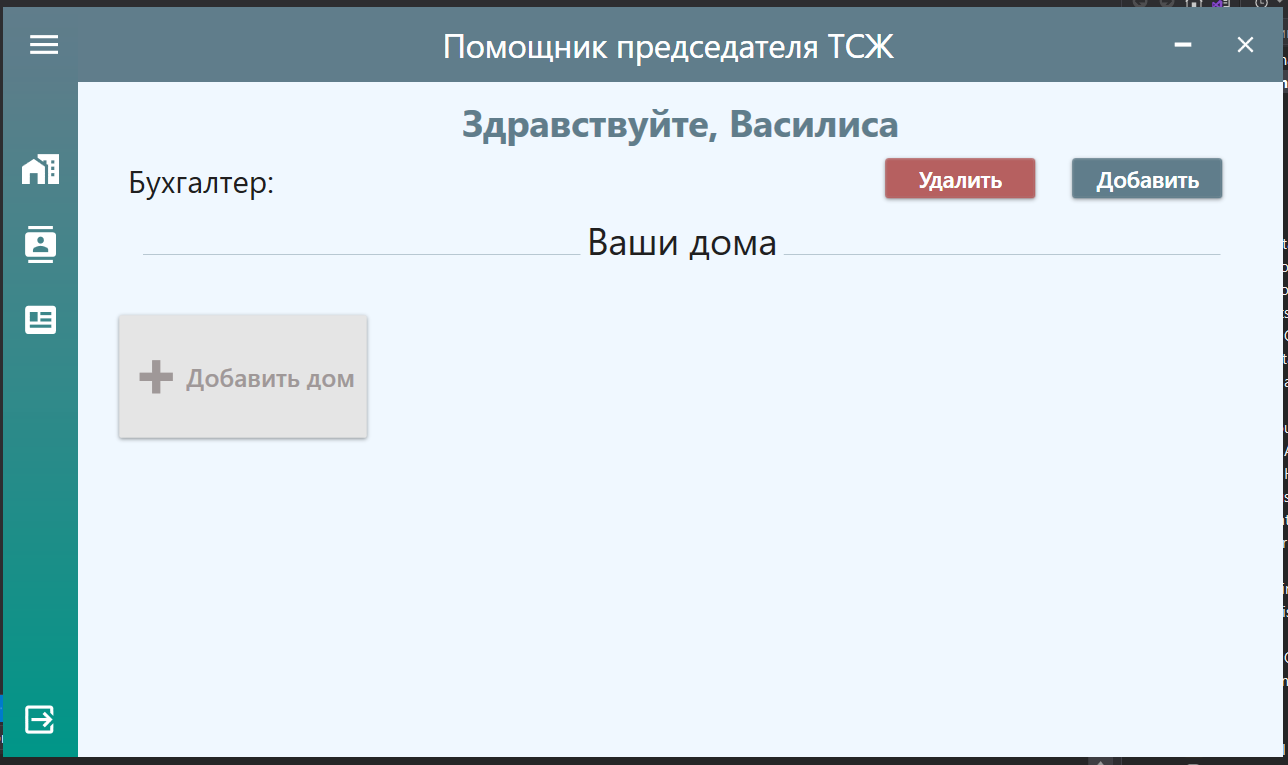


Рисунок 7.2 – Главная страница нового пользователя

Далее пользователь может добавить дом.

На рисунке 7.3 изображена страница добавления дома и информации о нем.

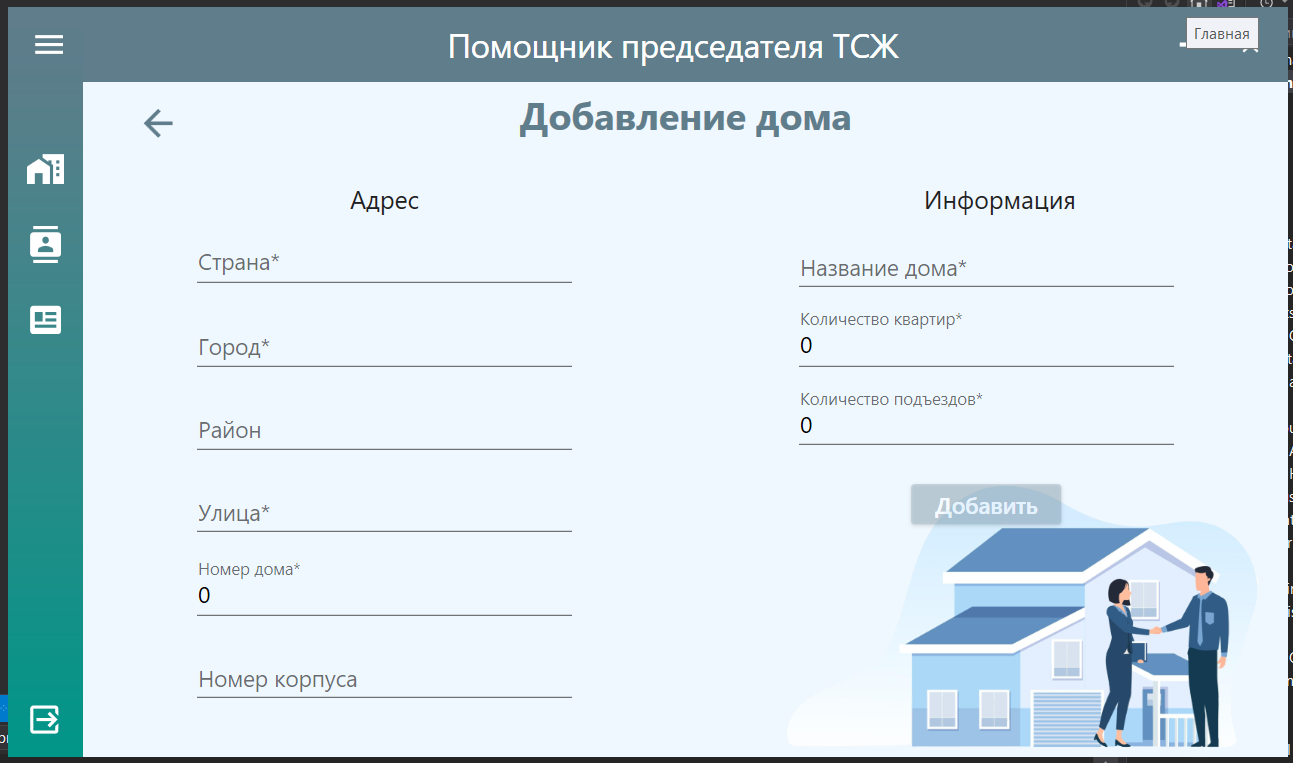


Рисунок 7.3 – Страница добавления дома

После добавления дома на главной странице пользователя появится ранее добавленный дом (рисунок 7.4).

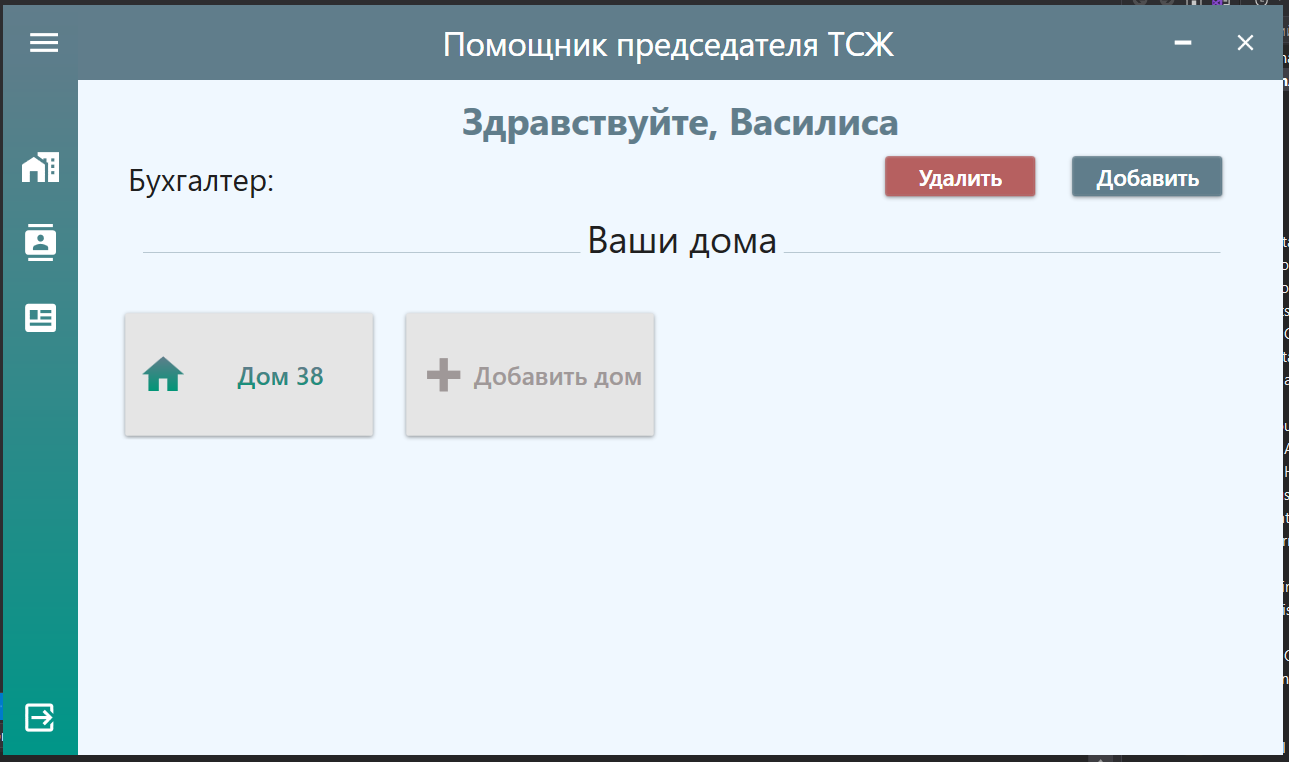


Рисунок 7.4 – Главная страница после добавление дома

Так как у председателя в управлении может находится несколько домов, приложение автоматически создает прокрутку списка домов при необходимости.

Затем, можно добавить бухгалтера по нажатию кнопки «Добавить» напротив строки с надписью «Бухгалтер» (рисунок 7.5).

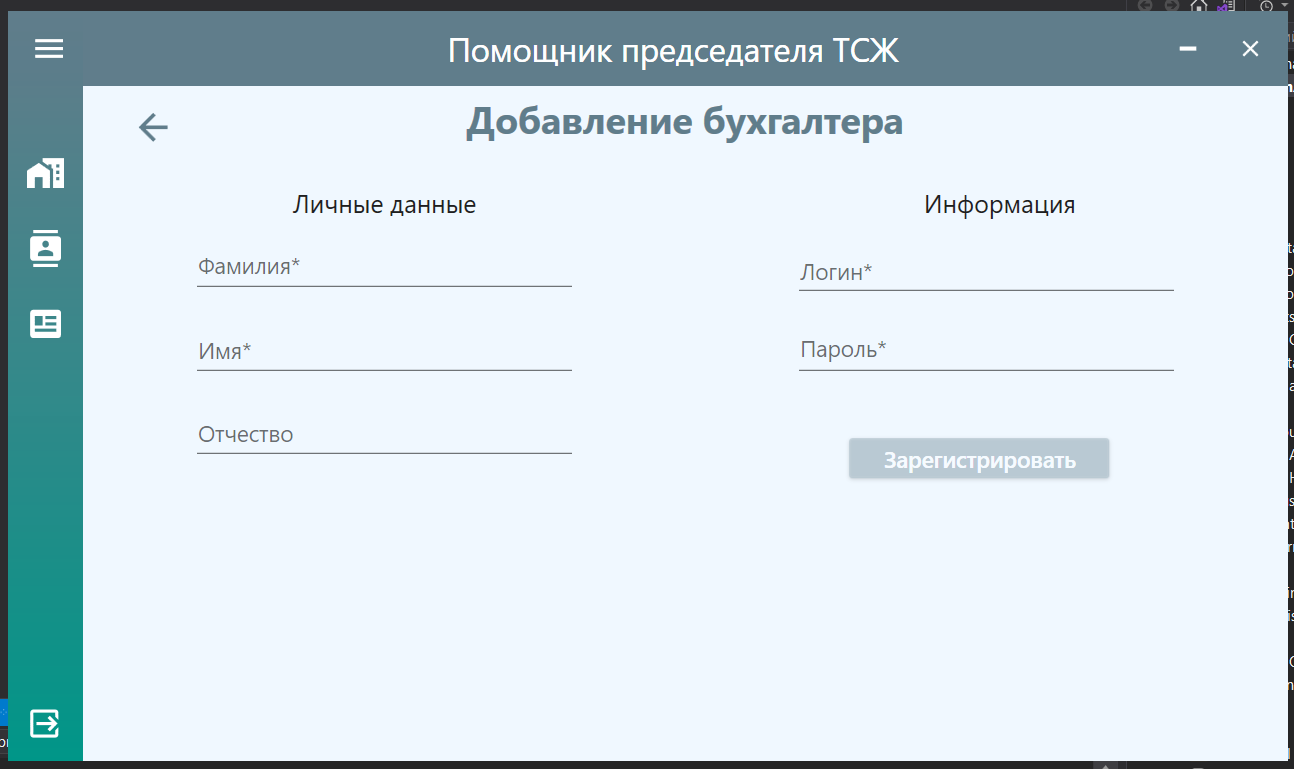


Рисунок 7.5 – Страница добавления бухгалтера

После добавления бухгалтера для данного дома заполняется поле на главной странице приложения, какой бухгалтер закреплен за председателем (рисунок 7.6).

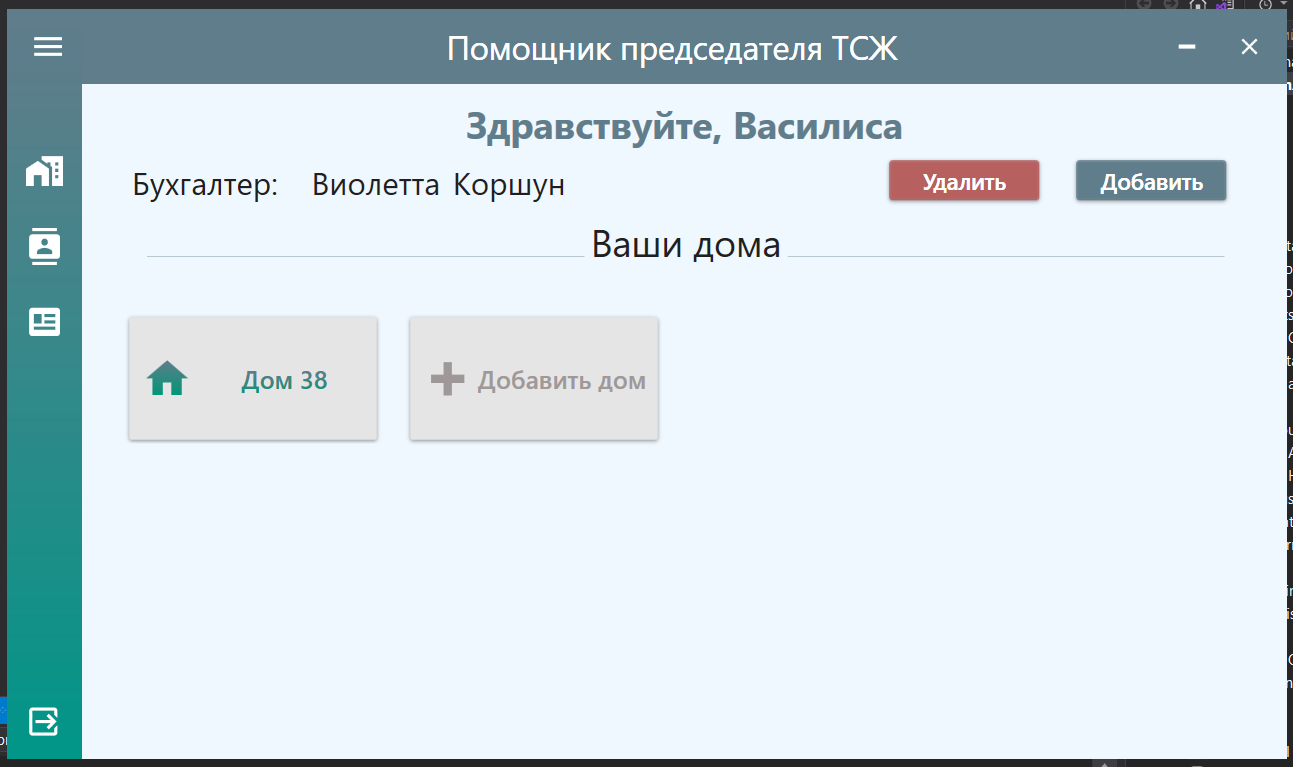


Рисунок 7.6 – Страница домов после добавления бухгалтера

При нажатии на кнопку с названием дома можно просмотреть информацию о нем: количество подъездов, квартиры, распределенные по вкладкам подъездов (рисунок 7.7).

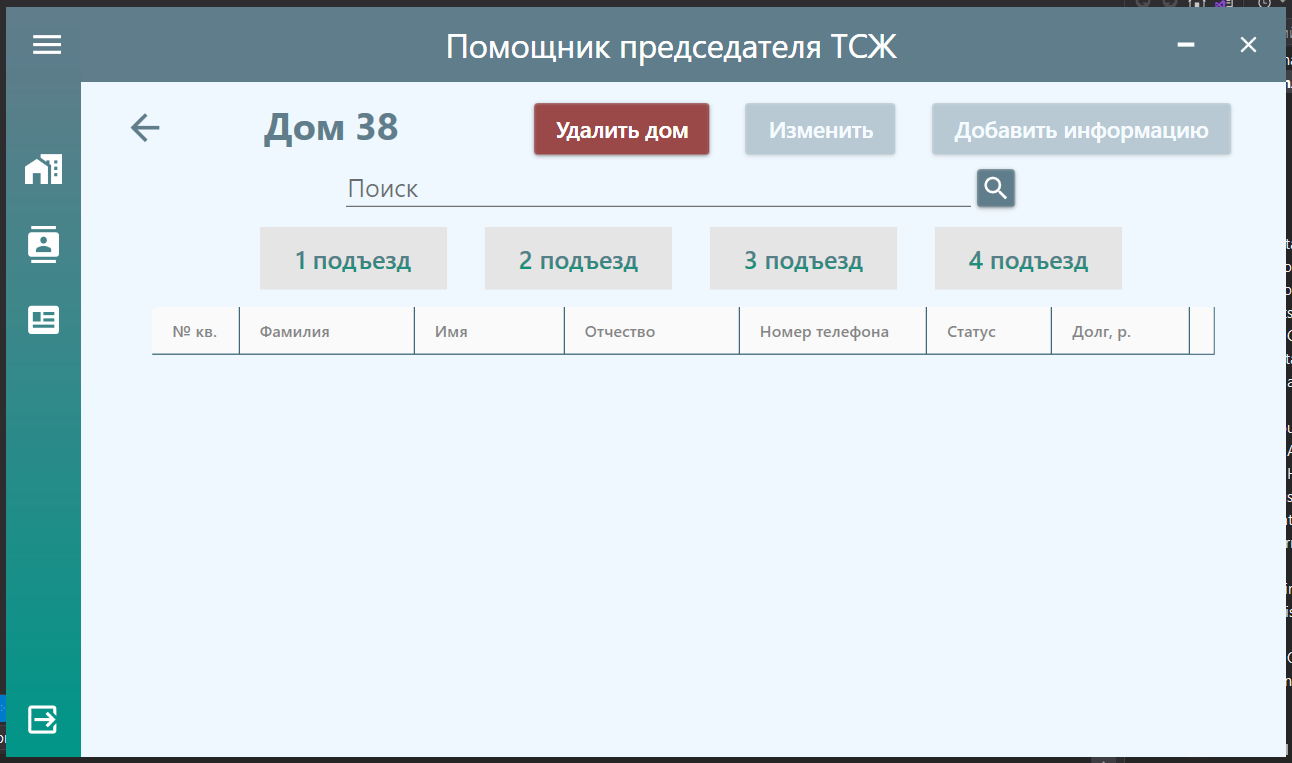


Рисунок 7.7 – Страница просмотра квартир дома

Также на данной странице можно удалить дом двойным нажатием на кнопку.

По нажатию на вкладку «Подъезд», автоматически генерируется запрос в базу данных для поиска подъезда в определенном доме. Пользователю представляются все доступные квартиры в доме.

При нажатии на строку с данными появляется возможность добавить жильца под определенным номером квартиры (рисунок 7.8).

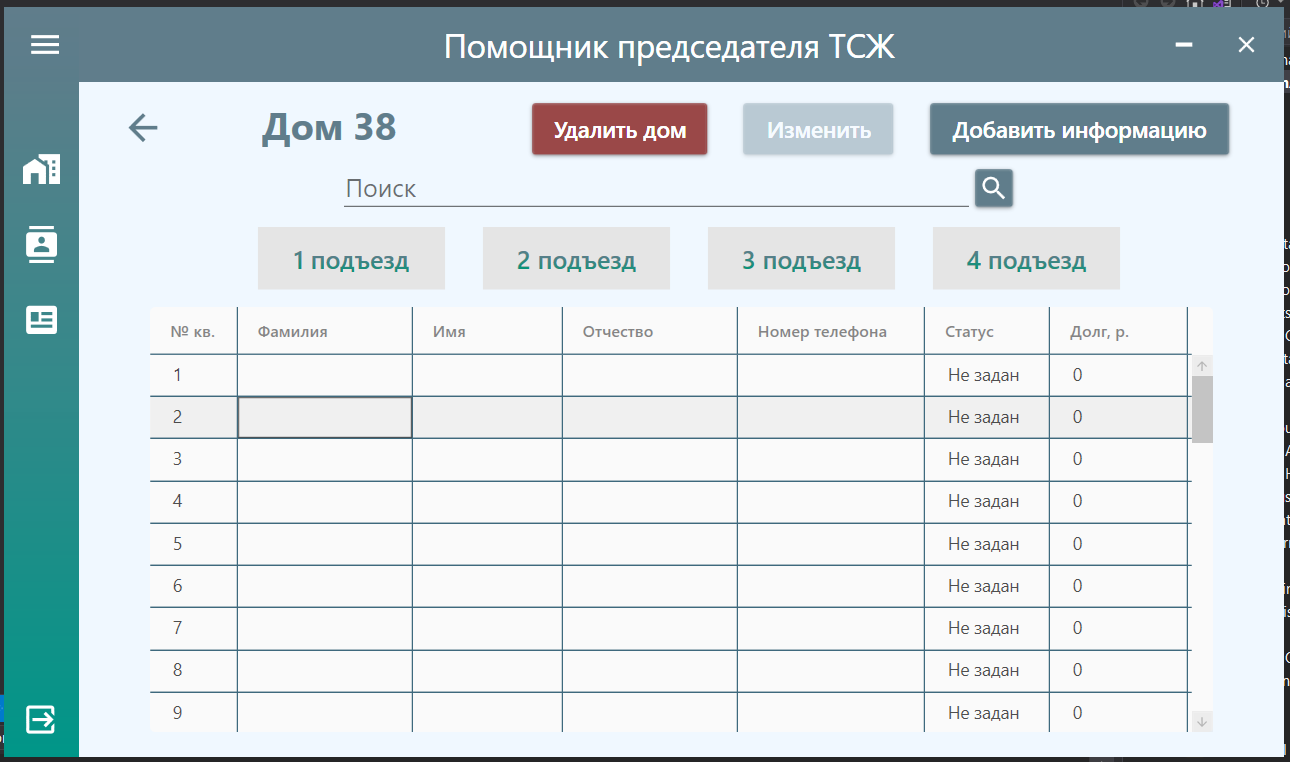


Рисунок 7.8 – Активация кнопки «Добавить информацию»

Добавление собственника осуществляется таким же образом, как и осуществляется добавление дома и бухгалтера (рисунок 7.9).

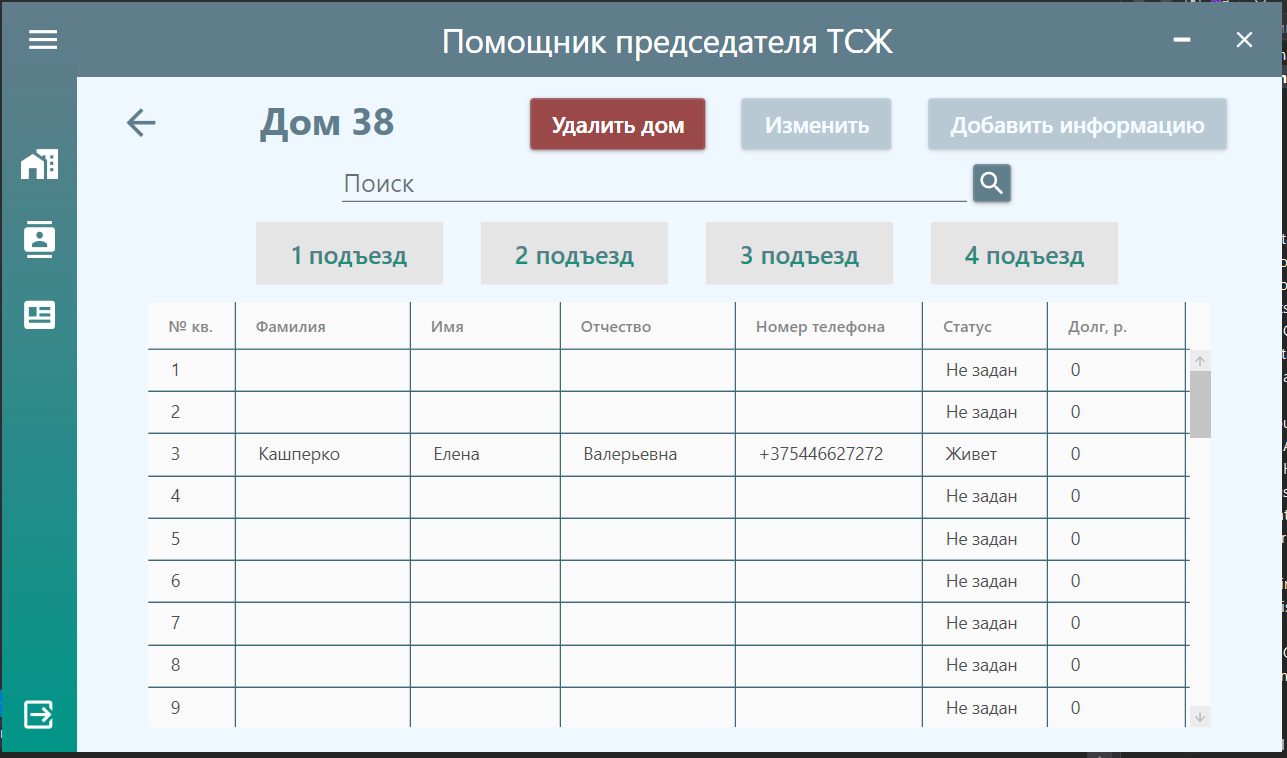


Рисунок 7.9 – Добавленный собственник в таблице

Таким образом пользователем заполняется весь журнал собственников.

При нажатии на уже заполненную строку, подсвечивается лишь кнопка «Изменить». Нажав ее, можно изменить информацию о выбранном собственнике или удалить его (рисунок 7.10).

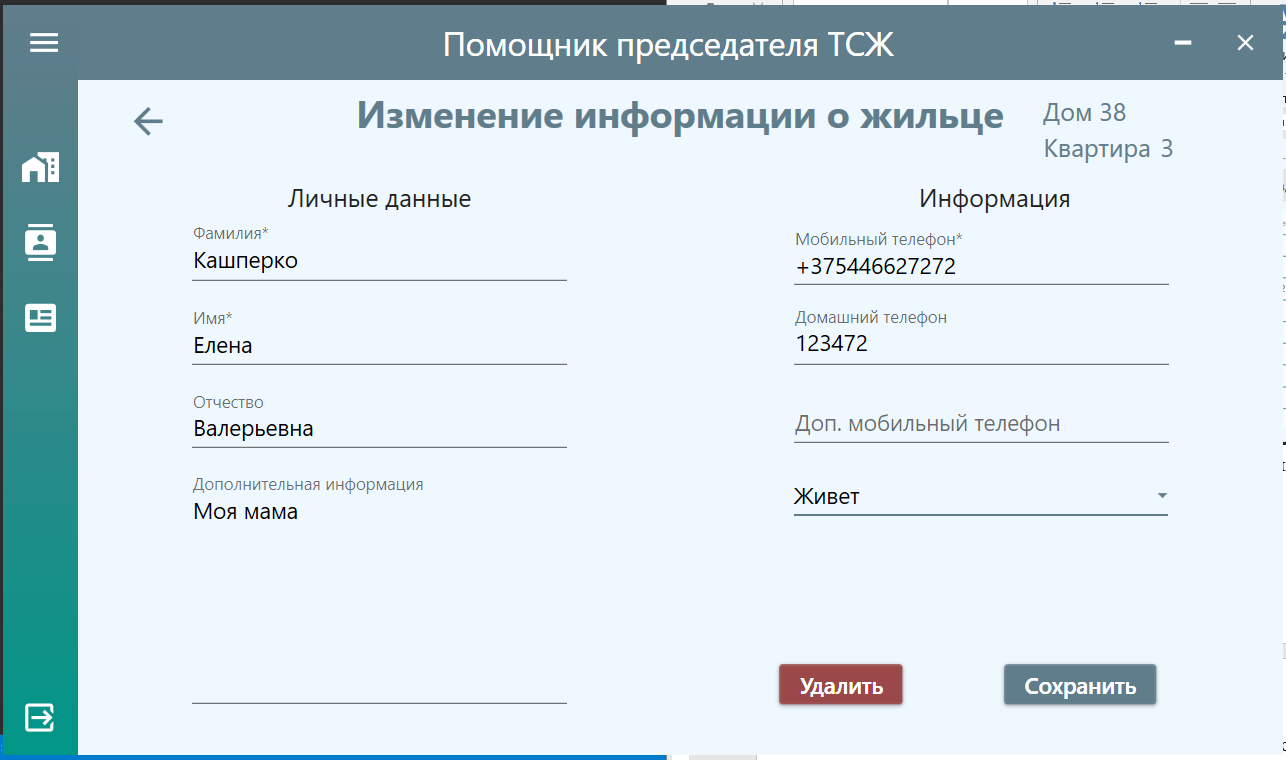


Рисунок 7.10 – Изменение информации о жильце собственнике

Перейдя к разделу «Записная книжка», если она не заполнена, она будет выглядеть так, как изображено на рисунке 7.11.

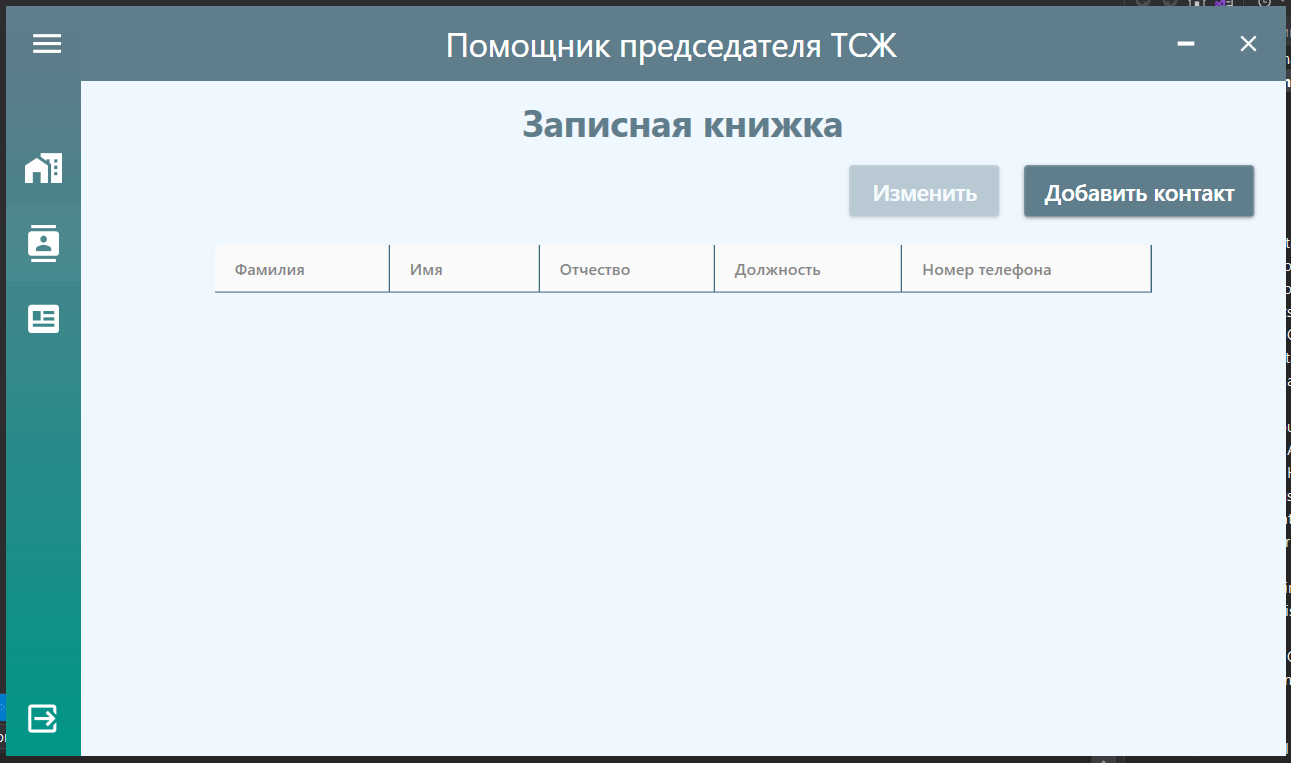


Рисунок 7.11 – Страница «Записная книжка», если в ней нет номеров

На рисунке выше можно заметить, что доступна кнопка «Добавить контакт».

Аналогично добавлению дома или жильца, заполняется форма (рисунок 7.12).

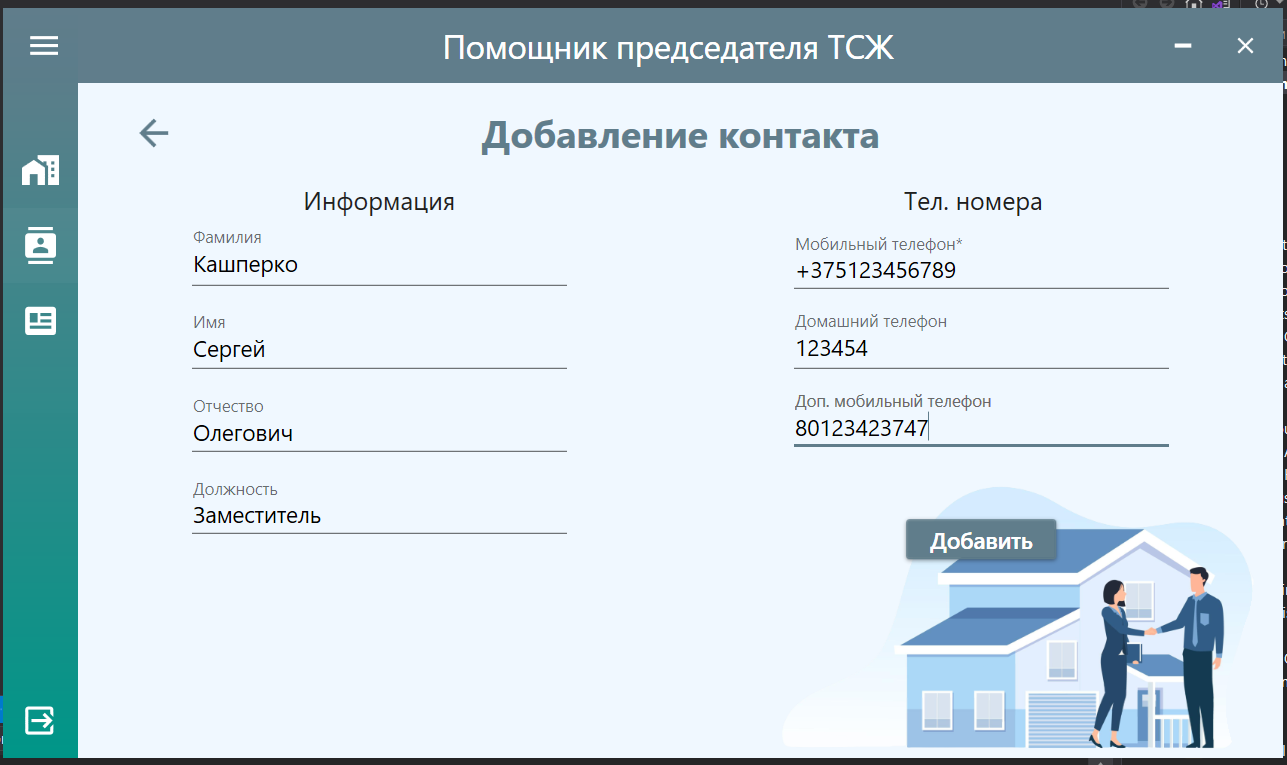


Рисунок 7.12 – Страница добавления контакта

После добавления контакта в записную книжку при нажатии на него, можно изменить контактную информацию (рисунок 7.13).

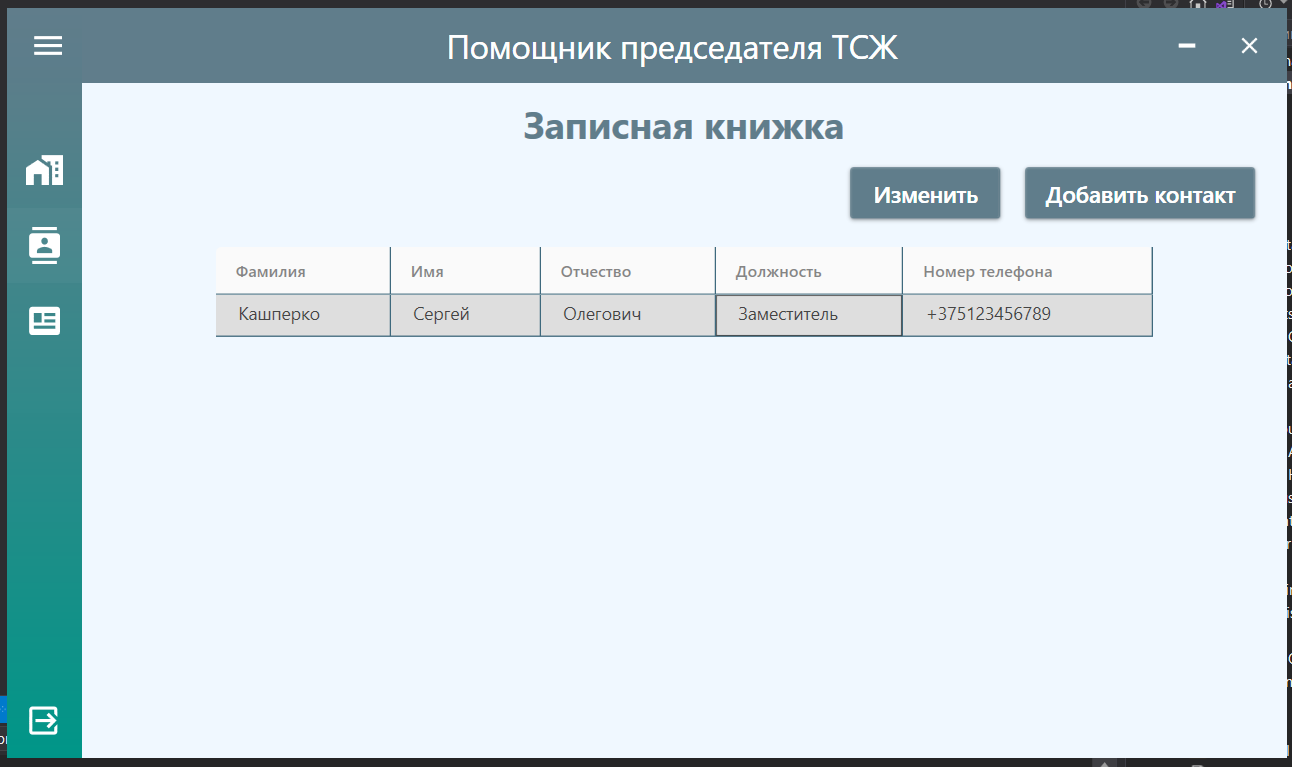


Рисунок 7.13 – Страница «Записная книжка» при выделении контакта

Таким образом пользователь с ролью «Председатель» может добавлять и изменять контакты в своей записной книжке.

При нажатии на раздел «Справка» открывается страница с важной информацией для председателя, представленная на рисунке 7.14.

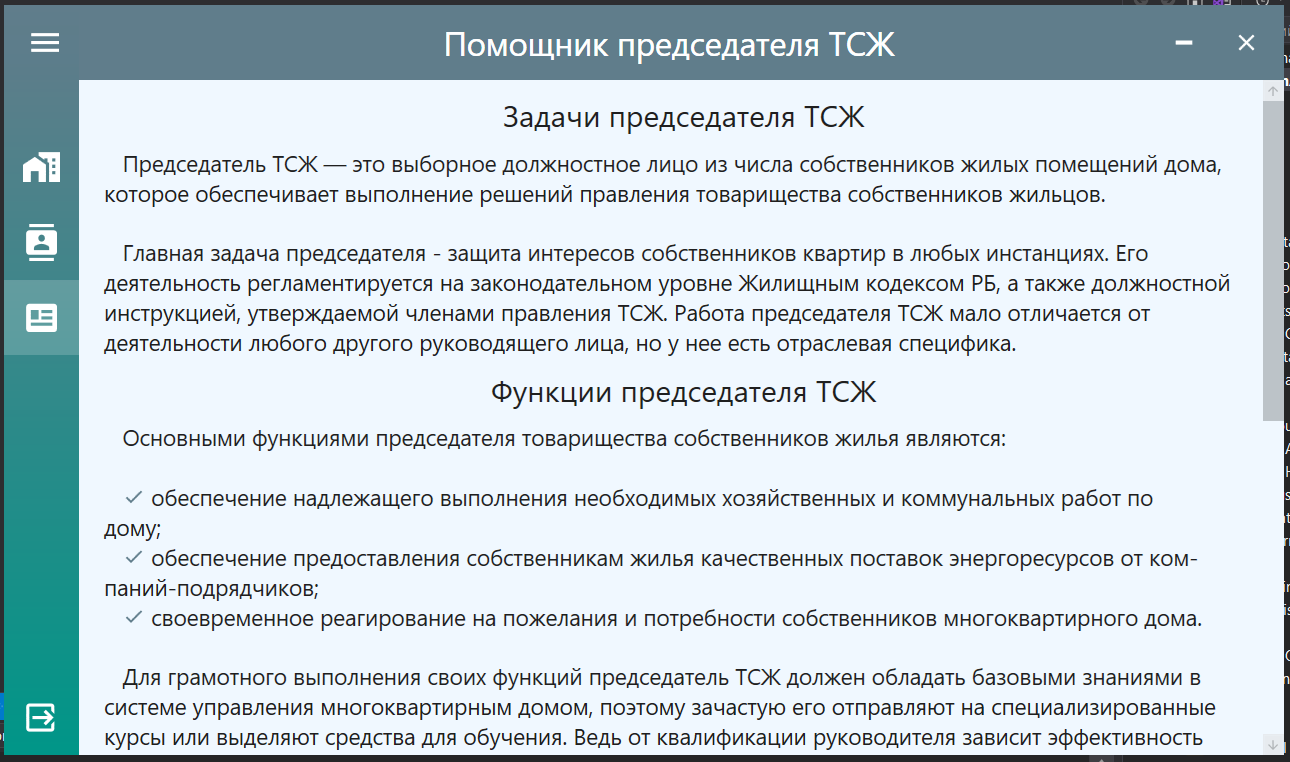


Рисунок 7.14 – Страница «Справка»

В том случае, когда бухгалтер авторизуется и заходит на свой аккаунт, для него отображаются только те дома, которые создал председатель, за которым закреплен сам бухгалтер (рисунок 7.15).

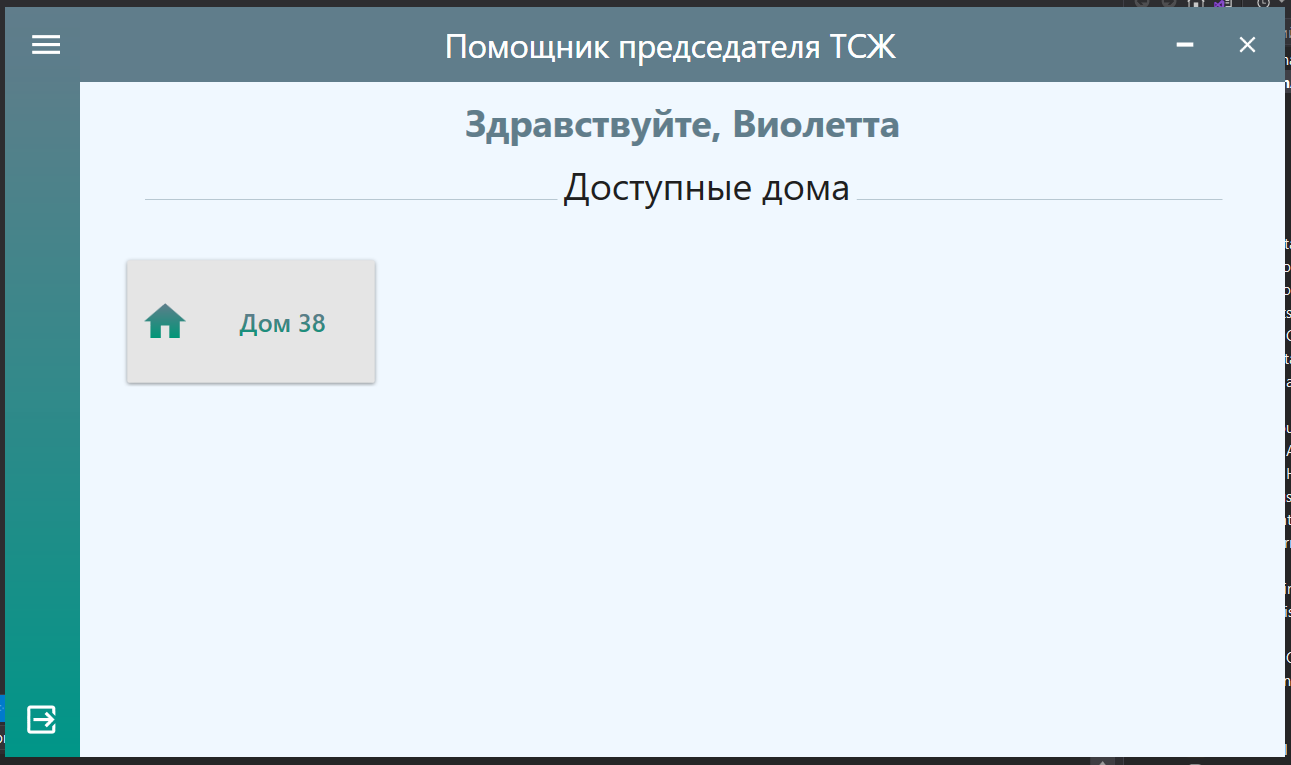


Рисунок 7.15 – Главная страница пользователя с ролью «Бухгалтер»

Таким образом функционал приложения бля пользователя с ролью «Бухгалтер» ограничен.

Нажав на кнопку, бухгалтер может перейти на страницу, на которой сможет только менять данные о жильцах, в том числе и их текущую задолженность, в отличии от председателя (рисунок 7.16).

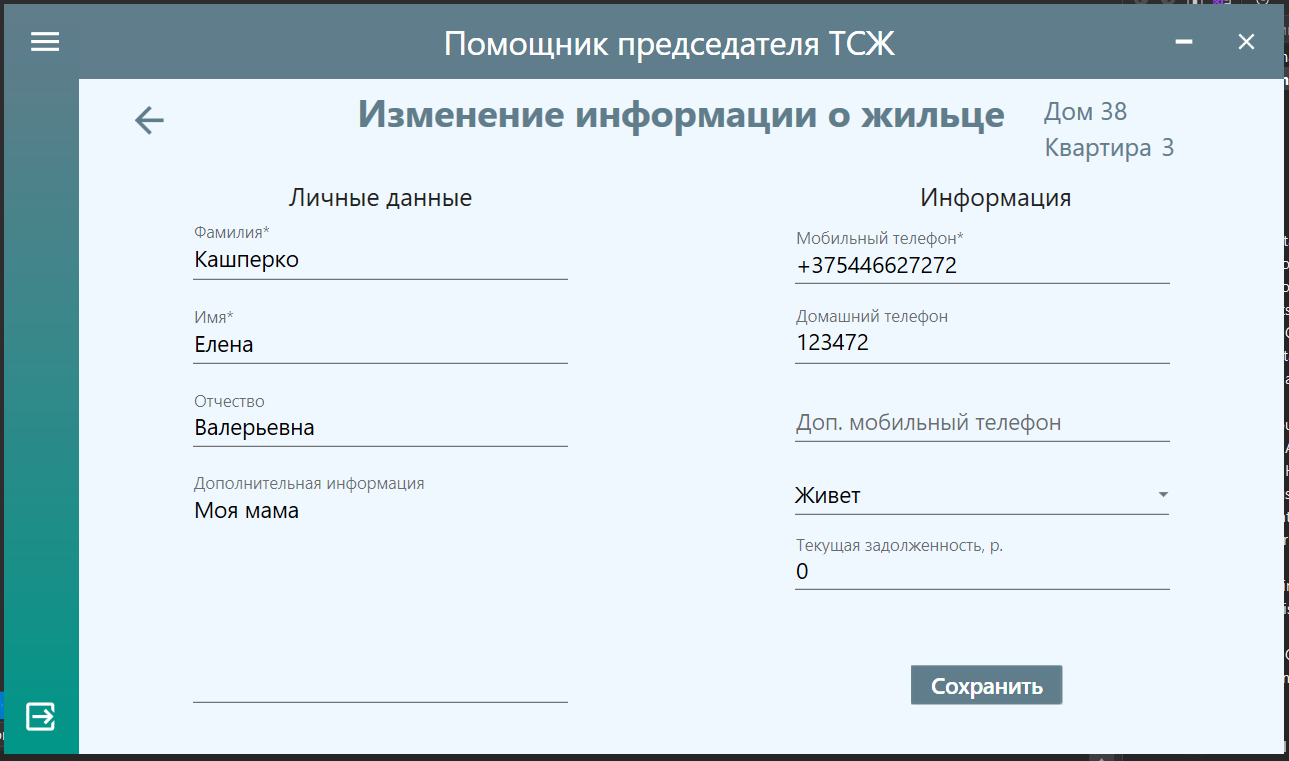


Рисунок 7.16 – Страница «Изменение информации о жильце» для бухгалтера

Исходя из проделанной работы мы можем отметить, что данная база данных правильно выполняет свои функции такие как защита от некорректного ввода данных, вывод необходимой информации, изменение строк, удаление и создание новых значений.

# ЗАКЛЮЧЕНИЕ

В процессе решения поставленной задачи была достигнута поставленная цель по разработке базы данных и созданию программного средства «Помощник председателя товарищества собственников жилья».

В данном проекте использовалась СУБД Oracle DataBase 12c.

Основной целью курсового проекта стало проектирование базы данных для дальнейшей интеграции с приложением, которое помогло бы облегчить взаимодействие с базой данных посредством программного интерфейса.

При разработке были выполнены все пункты из указанного списка предполагаемого основного функционала приложения.

В программном средстве были реализованы следующие функции председателя ТСЖ:

* + выполнять регистрацию и авторизацию;
  + поддерживать работу c базой данных;
  + добавлять и изменять информацию о жилых домах и собственниках квартир;
  + просматривать информацию о жилых домах и собственниках квартир;
  + выполнять поисковые запросы;
  + добавлять контакты работников дома и организаций в записную книжку;
  + выполнять регистрацию бухгалтера для определенного дома.

Функции бухгалтера ТСЖ:

* + выполнять авторизацию;
  + просматривать информацию о собственниках квартир;
  + добавлять и обновлять сведения о текущих задолженностях собственников.

База данных прошла тестирование при использовании в БД большого количество данных. Также были реализованы процедуры для импорта, экспорта данных в формат XML.

Была реализована технология «Резервное копирование и восстановление».

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная база данных работает верно, а требования технического задания выполнены в полном объёме.

# Cписок используемых источников

1. Официальный сайт Oracle [Электронный ресурс]// Oracle. – Режим доступа: <https://www.oracle.com/database/database-vault/index.html>. – Дата доступа: 08.03.2023.
2. Export and Import In Oracle XML DB Data [Электронный ресурс]// Oracle XML. – Режим доступа: <https://docs.oracle.com/database/121/ADXDB/xdb26imp.htm#AD>  
   XDB3000. – Дата доступа: 02.04.2023.
3. Getting Started with RMAN [Электронный ресурс]// RMAN. – Режим доступа: [https://docs.oracle.com/cd/E11882\_01/backup.112/e10642/rcmquick.htm#BRADV893  
   46](https://docs.oracle.com/cd/E11882_01/backup.112/e10642/rcmquick.htm#BRADV89346). – Дата доступа: 24.04.2023.
4. ProfessorWeb .NET & Web Programming [Электронный ресурс]// ProfessorWeb. – Режим доступа: <https://professorweb.ru>. – Дата доступа: 26.04.2023.
5. Microsoft Docs Archived Content [Электронный ресурс]// Microsoft Docs. – Режим доступа: <https://docs.microsoft.com/en-us/archive/>. – Дата доступа: 26.04.2023.
6. Форум для программистов или разработчиков [Электронный ресурс]// StackOverflow. – Режим доступа: <https://stackoverflow.com/> – Дата доступа: 28.04.2023.

# ПРИЛОЖЕНИЕ А

Скрипт создания таблиц и ограничений целостности

|  |
| --- |
| -------------- Таблицы --------------  --DROP TABLE Addresses;  --DROP TABLE Contacts;  --DROP TABLE Flats;  --DROP TABLE Houses;  --DROP TABLE Owners;  --DROP TABLE PhoneNumbers;  --DROP TABLE Porches;  --DROP TABLE Users;  --1. Таблица "Addresses"  CREATE TABLE Addresses (  AddressId NUMBER GENERATED BY DEFAULT AS IDENTITY NOT NULL,  Country NVARCHAR2(20) NOT NULL,  City NVARCHAR2(20) NOT NULL,  District NVARCHAR2(30),  Street NVARCHAR2(30) NOT NULL,  HouseNumber NUMBER NOT NULL,  HousingNumber NVARCHAR2(5),  CONSTRAINT PK\_Addresses PRIMARY KEY (AddressId)  );  --2. Таблица "Contacts"  CREATE TABLE Contacts (  ContactId NUMBER GENERATED BY DEFAULT AS IDENTITY NOT NULL,  Surname NVARCHAR2(30),  Name NVARCHAR2(20),  Patronymic NVARCHAR2(20),  Position NVARCHAR2(20) NOT NULL,  PhoneNumberId NUMBER NOT NULL,  UserId NUMBER NOT NULL,  CONSTRAINT PK\_Contacts PRIMARY KEY (ContactId)  );  --3. Таблица "Flats"  CREATE TABLE Flats (  FlatId NUMBER GENERATED BY DEFAULT AS IDENTITY NOT NULL,  FlatNumber NUMBER NOT NULL,  PorchId NUMBER NOT NULL,  CONSTRAINT PK\_Flats PRIMARY KEY (FlatId)  );  --4. Таблица "Houses"  CREATE TABLE Houses (  HouseId NUMBER GENERATED BY DEFAULT AS IDENTITY NOT NULL,  HouseName NVARCHAR2(20) NOT NULL,  NumberOfFlats NUMBER NOT NULL,  NumberOfPorches NUMBER NOT NULL,  AddressId NUMBER NOT NULL,  UserId NUMBER NOT NULL,  CONSTRAINT PK\_Houses PRIMARY KEY (HouseId)  );  --5. Таблица "Owners"  CREATE TABLE Owners (  OwnerId NUMBER GENERATED BY DEFAULT AS IDENTITY NOT NULL,  Surname NVARCHAR2(30) NOT NULL,  Name NVARCHAR2(20) NOT NULL,  Patronymic NVARCHAR2(20),  AdditionalInfo NVARCHAR2(30),  CurrentDebt NUMBER NOT NULL,  PhoneNumberId NUMBER NOT NULL,  OwnerStatusId NUMBER NOT NULL,  FlatId NUMBER NOT NULL,  CONSTRAINT PK\_Owners PRIMARY KEY (OwnerId)  );  --6. Таблица "PhoneNumbers"  CREATE TABLE PhoneNumbers (  PhoneNumberId NUMBER GENERATED BY DEFAULT AS IDENTITY START WITH 1 NOT NULL,  MobilePhone NVARCHAR2(20),  HomePhone NVARCHAR2(20),  AdditionalPhone NVARCHAR2(20),  CONSTRAINT PK\_PhoneNumbers PRIMARY KEY (PhoneNumberId)  );  --7. Таблица "Porches"  CREATE TABLE Porches(  PorchId INTEGER GENERATED BY DEFAULT AS IDENTITY NOT NULL,  PorchNumber INTEGER NOT NULL,  HouseId INTEGER NOT NULL,  CONSTRAINT PK\_Porches PRIMARY KEY (PorchId)  );  --8. Таблица "Users"  CREATE TABLE Users (  UserId NUMBER GENERATED BY DEFAULT AS IDENTITY NOT NULL,  Surname NVARCHAR2(30) NOT NULL,  Name NVARCHAR2(20) NOT NULL,  Patronymic NVARCHAR2(20),  Login NVARCHAR2(20) NOT NULL UNIQUE,  Password NVARCHAR2(40) NOT NULL,  Role NUMBER NOT NULL,  AccountantId NUMBER,  CONSTRAINT PK\_Users PRIMARY KEY (UserId)  );  -------------- Ограничения целостности --------------  --ALTER TABLE Contacts DROP CONSTRAINT PhoneNumbers\_PhoneNumberId;  --ALTER TABLE Contacts DROP CONSTRAINT Users\_UserId;  --ALTER TABLE Flats DROP CONSTRAINT Porches\_PorchId;  --ALTER TABLE Houses DROP CONSTRAINT Addresses\_AddressId;  --ALTER TABLE Houses DROP CONSTRAINT Houses\_UserId;  --ALTER TABLE Houses DROP CONSTRAINT UQ\_Houses\_UserId\_AddressId;  --ALTER TABLE Owners DROP CONSTRAINT Flats\_FlatId;  --ALTER TABLE Owners DROP CONSTRAINT Owners\_PhoneNumberId;  --ALTER TABLE Porches DROP CONSTRAINT Houses\_HouseId;  ALTER TABLE Contacts ADD CONSTRAINT PhoneNumbers\_PhoneNumberId FOREIGN KEY (PhoneNumberId) REFERENCES PhoneNumbers(PhoneNumberId) ON DELETE CASCADE;  ALTER TABLE Contacts ADD CONSTRAINT Users\_UserId FOREIGN KEY (UserId) REFERENCES Users(UserId) ON DELETE CASCADE;  ALTER TABLE Flats ADD CONSTRAINT Porches\_PorchId FOREIGN KEY (PorchId) REFERENCES Porches(PorchId) ON DELETE CASCADE;  ALTER TABLE Houses ADD CONSTRAINT Addresses\_AddressId FOREIGN KEY (AddressId) REFERENCES Addresses(AddressId) ON DELETE CASCADE;  ALTER TABLE Houses ADD CONSTRAINT Houses\_UserId FOREIGN KEY (UserId) REFERENCES Users(UserId) ON DELETE CASCADE;  ALTER TABLE Houses ADD CONSTRAINT UQ\_Houses\_UserId\_AddressId UNIQUE (UserId, AddressId);  ALTER TABLE Owners ADD CONSTRAINT Flats\_FlatId FOREIGN KEY (FlatId) REFERENCES Flats(FlatId) ON DELETE CASCADE;  ALTER TABLE Owners ADD CONSTRAINT Owners\_PhoneNumberId FOREIGN KEY (PhoneNumberId) REFERENCES PhoneNumbers(PhoneNumberId) ON DELETE CASCADE;  ALTER TABLE Porches ADD CONSTRAINT Houses\_HouseId FOREIGN KEY (HouseId) REFERENCES Houses(HouseId) ON DELETE CASCADE; |

# ПРИЛОЖЕНИЕ Б

Скрипт создания ролей и пользователей

|  |
| --- |
| -------------- Роль администратора --------------  --drop role C##Administrator;  --drop role C##Chairman;  --drop role C##Accountant;  create role C##Administrator;  GRANT ALL PRIVILEGES ON Addresses TO C##Administrator;  GRANT ALL PRIVILEGES ON Contacts TO C##Administrator;  GRANT ALL PRIVILEGES ON Flats TO C##Administrator;  GRANT ALL PRIVILEGES ON Houses TO C##Administrator;  GRANT ALL PRIVILEGES ON Owners TO C##Administrator;  GRANT ALL PRIVILEGES ON PhoneNumbers TO C##Administrator;  GRANT ALL PRIVILEGES ON Porches TO C##Administrator;  GRANT ALL PRIVILEGES ON Users TO C##Administrator;  GRANT CREATE SESSION TO C##Administrator;  GRANT DBA TO C##Administrator;  -------------- Роль председателя --------------  create role C##Chairman;  GRANT ALL PRIVILEGES ON C##Vasilisa.Houses TO C##Chairman;  GRANT ALL PRIVILEGES ON C##Vasilisa.Addresses TO C##Chairman;  GRANT ALL PRIVILEGES ON C##Vasilisa.Porches TO C##Chairman;  GRANT ALL PRIVILEGES ON C##Vasilisa.Flats TO C##Chairman;  GRANT ALL PRIVILEGES ON C##Vasilisa.Contacts TO C##Chairman;  GRANT ALL PRIVILEGES ON C##Vasilisa.Owners TO C##Chairman;  GRANT ALL PRIVILEGES ON C##Vasilisa.PhoneNumbers TO C##Chairman;  GRANT CREATE SESSION TO C##Chairman;  -------------- Роль бухгалетра --------------  create role C##Accountant;  GRANT SELECT ON C##Vasilisa.Houses TO C##Accountant;  GRANT SELECT ON C##Vasilisa.Addresses TO C##Accountant;  GRANT SELECT ON C##Vasilisa.Porches TO C##Accountant;  GRANT SELECT ON C##Vasilisa.Flats TO C##Accountant;  GRANT SELECT ON C##Vasilisa.PhoneNumbers TO C##Accountant;  GRANT ALL PRIVILEGES ON C##Vasilisa.Owners TO C##Accountant;  GRANT ALL PRIVILEGES ON C##Vasilisa.Contacts TO C##Accountant;  GRANT CREATE SESSION TO C##Accountant;  -------------- Пользователи --------------  --DROP USER C##Vasilisa CASCADE;  --DROP USER C##Youri;  --DROP USER C##Lisa;  CREATE USER C##Vasilisa IDENTIFIED BY Pa$$w0rd;  GRANT C##Administrator TO C##Vasilisa;  GRANT UNLIMITED TABLESPACE TO C##Vasilisa;  GRANT EXECUTE ANY PROCEDURE TO C##Vasilisa;  SET ROLE C##Administrator;  CREATE USER C##Youri IDENTIFIED BY Pa$$w0rd;  GRANT C##Chairman TO C##Youri;  GRANT UNLIMITED TABLESPACE TO C##Youri;  GRANT EXECUTE ANY PROCEDURE TO C##Youri;  SET ROLE C##Chairman;  CREATE USER C##Lisa IDENTIFIED BY Pa$$w0rd;  GRANT C##Accountant TO C##Lisa;  GRANT UNLIMITED TABLESPACE TO C##Lisa;  SET ROLE C##Accountant;  -------------- Просмотр пользователей --------------  SELECT username FROM all\_users;  SELECT \* FROM dba\_role\_privs;  SELECT DISTINCT TABLE\_NAME FROM DBA\_TAB\_PRIVS WHERE GRANTEE = 'C##ACCOUNTANT';  SELECT username FROM dba\_users WHERE account\_status = 'OPEN';  -------------- Проверка таблиц --------------  SELECT \* FROM C##Vasilisa.addresses; |

# ПРИЛОЖЕНИЕ В

Скрипт создания процедур

|  |
| --- |
| -------------- Добавление пользователя --------------  CREATE OR REPLACE PROCEDURE InsertUser(  p\_Surname IN NVARCHAR2,  p\_Name IN NVARCHAR2,  p\_Patronymic IN NVARCHAR2,  p\_Login IN NVARCHAR2,  p\_Password IN NVARCHAR2,  p\_Role IN NUMBER,  p\_AccountantId IN NUMBER  )  IS  BEGIN  INSERT INTO C##Vasilisa.Users (Surname, Name, Patronymic, Login, Password, Role, AccountantId)  VALUES (p\_Surname, p\_Name, p\_Patronymic, p\_Login, p\_Password, p\_Role, p\_AccountantId);    COMMIT;    DBMS\_OUTPUT.PUT\_LINE('User inserted successfully.');  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('User with the same login already exists.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error inserting user: ' || SQLERRM);  RAISE;  END;  BEGIN  C##Vasilisa.InsertUser('Кашперко', 'Василиса', 'Сергеевна', 'vasilisa1', 'vasilisa1', 0, 0);  END;  BEGIN  C##Vasilisa.InsertUser('Кашперко', 'Сергей', 'Олегович', 'sergey1', 'sergey1', 1, 1);  END;  -------------- Изменение данных пользователя --------------  CREATE OR REPLACE PROCEDURE AlterUser(  p\_UserId IN NUMBER,  p\_Surname IN NVARCHAR2,  p\_Name IN NVARCHAR2,  p\_Patronymic IN NVARCHAR2,  p\_Login IN NVARCHAR2,  p\_Password IN NVARCHAR2,  p\_Role IN NUMBER,  p\_AccountantId IN NUMBER  )  IS  BEGIN  UPDATE C##Vasilisa.Users  SET Surname = p\_Surname,  Name = p\_Name,  Patronymic = p\_Patronymic,  Login = p\_Login,  Password = p\_Password,  Role = p\_Role,  AccountantId = p\_AccountantId  WHERE UserId = p\_UserId;    IF SQL%ROWCOUNT = 0 THEN  RAISE NO\_DATA\_FOUND;  END IF;    COMMIT;    DBMS\_OUTPUT.PUT\_LINE('User altered successfully.');  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('User with ID ' || p\_UserId || ' not found.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error altering user: ' || SQLERRM);  END;  BEGIN  C##Vasilisa.AlterUser(3, 'Кашперко', 'Сергей', 'Олегович', 'sergey1', 'sergey1', 1, 2);  END;  -------------- Удаление пользователя --------------  CREATE OR REPLACE PROCEDURE DeleteUser(  p\_UserId IN NUMBER  )  IS  BEGIN  DELETE FROM C##Vasilisa.Users  WHERE UserId = p\_UserId;    IF SQL%ROWCOUNT = 0 THEN  RAISE NO\_DATA\_FOUND;  END IF;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('User deleted successfully.');  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('User with ID ' || p\_UserId || ' not found.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleting user: ' || SQLERRM);  END;  BEGIN  C##Vasilisa.DeleteUser(2);  END;  -------------- Удаление всех пользователей --------------  CREATE OR REPLACE PROCEDURE DeleteAllUsers  IS  BEGIN  BEGIN  DELETE FROM C##Vasilisa.Users;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('No users found.');  ROLLBACK;  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('All users deleted successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleting users: ' || SQLERRM);  END;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error deleting users: ' || SQLERRM);  END;  BEGIN  DeleteAllUsers;  END;  -------------- Найти пользователя по фамилии --------------  CREATE OR REPLACE PROCEDURE FindUserBySurname(  p\_Surname IN NVARCHAR2  )  IS  CURSOR c\_Users IS  SELECT \*  FROM C##Vasilisa.Users  WHERE Surname = p\_Surname;    v\_User Users%ROWTYPE;  v\_UserFound BOOLEAN := FALSE;  BEGIN  DBMS\_OUTPUT.PUT\_LINE('-----------------------');  DBMS\_OUTPUT.PUT\_LINE('Founded user(s):');  DBMS\_OUTPUT.PUT\_LINE('-----------------------');  OPEN c\_Users;    LOOP  FETCH c\_Users INTO v\_User;  EXIT WHEN c\_Users%NOTFOUND;    v\_UserFound := TRUE;    DBMS\_OUTPUT.PUT\_LINE('User ID: ' || v\_User.UserId);  DBMS\_OUTPUT.PUT\_LINE('Name: ' || v\_User.Name);  DBMS\_OUTPUT.PUT\_LINE('Surname: ' || v\_User.Surname);    DBMS\_OUTPUT.PUT\_LINE('-----------------------');  END LOOP;    CLOSE c\_Users;    IF NOT v\_UserFound THEN  DBMS\_OUTPUT.PUT\_LINE('No user found with the provided surname: ' || p\_Surname);  END IF;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error finding user: ' || SQLERRM);  END;  BEGIN  C##Vasilisa.FindUserBySurname('Doe');  END;  -------------- Найти пользователя по идентификатору --------------  CREATE OR REPLACE PROCEDURE FindUserById(  p\_UserId IN NUMBER  )  IS  CURSOR c\_Users (p\_UserId NUMBER) IS  SELECT \*  FROM C##Vasilisa.Users  WHERE UserId = p\_UserId;    v\_User Users%ROWTYPE;  BEGIN  DBMS\_OUTPUT.PUT\_LINE('-----------------------');  DBMS\_OUTPUT.PUT\_LINE('Founded user:');  DBMS\_OUTPUT.PUT\_LINE('-----------------------');    OPEN c\_Users(p\_UserId);    FETCH c\_Users INTO v\_User;    IF c\_Users%FOUND THEN  LOOP  DBMS\_OUTPUT.PUT\_LINE('User ID: ' || v\_User.UserId);  DBMS\_OUTPUT.PUT\_LINE('Name: ' || v\_User.Name);  DBMS\_OUTPUT.PUT\_LINE('Surname: ' || v\_User.Surname);    DBMS\_OUTPUT.PUT\_LINE('-----------------------');    FETCH c\_Users INTO v\_User;  EXIT WHEN c\_Users%NOTFOUND;  END LOOP;  ELSE  DBMS\_OUTPUT.PUT\_LINE('No user found with the provided ID: ' || p\_UserId);  END IF;    CLOSE c\_Users;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('No user found with the provided ID: ' || p\_UserId);  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error finding user: ' || SQLERRM);  END;  BEGIN  C##Vasilisa.FindUserById(1);  END;  -------------- Вывод всех пользователей --------------  SELECT \* FROM USERS order by UserId;  -------------- Добавление адреса --------------  CREATE OR REPLACE PROCEDURE InsertAddress(  p\_Country IN NVARCHAR2,  p\_City IN NVARCHAR2,  p\_District IN NVARCHAR2,  p\_Street IN NVARCHAR2,  p\_HouseNumber IN NUMBER,  p\_HousingNumber IN NVARCHAR2  )  AS  BEGIN  INSERT INTO C##Vasilisa.Addresses (Country, City, District, Street, HouseNumber, HousingNumber)  VALUES (p\_Country, p\_City, p\_District, p\_Street, p\_HouseNumber, p\_HousingNumber);    COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Address inserted successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error occurred while inserting the address.');  DBMS\_OUTPUT.PUT\_LINE(SQLERRM);  END;  BEGIN  C##Vasilisa.InsertAddress('Country1', 'City1', 'District1', 'Street1', 123, 'A');  END;  -------------- Изменение данных адреса --------------  CREATE OR REPLACE PROCEDURE AlterAddress(  p\_AddressId IN NUMBER,  p\_Country IN NVARCHAR2,  p\_City IN NVARCHAR2,  p\_District IN NVARCHAR2,  p\_Street IN NVARCHAR2,  p\_HouseNumber IN NUMBER,  p\_HousingNumber IN NVARCHAR2  )  AS  v\_Count NUMBER;  BEGIN  SELECT COUNT(\*)  INTO v\_Count  FROM C##Vasilisa.Addresses  WHERE AddressId = p\_AddressId;  IF v\_Count = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('Address with AddressId ' || p\_AddressId || ' not found.');  RETURN;  END IF;  UPDATE C##Vasilisa.Addresses  SET Country = p\_Country,  City = p\_City,  District = p\_District,  Street = p\_Street,  HouseNumber = p\_HouseNumber,  HousingNumber = p\_HousingNumber  WHERE AddressId = p\_AddressId;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Address updated successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error occurred while updating the address.');  DBMS\_OUTPUT.PUT\_LINE(SQLERRM);  END;  BEGIN  C##Vasilisa.AlterAddress(1, 'NewCountry', 'NewCity', 'NewDistrict', 'NewStreet', 456, 'B');  END;  -------------- Удаление адреса --------------  CREATE OR REPLACE PROCEDURE DeleteAddress(  p\_AddressId IN NUMBER  )  IS  BEGIN  DELETE FROM C##Vasilisa.Addresses  WHERE AddressId = p\_AddressId;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('Address with ID ' || p\_AddressId || ' not found.');  ROLLBACK;  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Address deleted successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleting address: ' || SQLERRM);  RAISE;  END;  BEGIN  C##Vasilisa.DeleteAddress(1);  END;  -------------- Удаление всех адресов --------------  CREATE OR REPLACE PROCEDURE DeleteAllAddresses  IS  BEGIN  BEGIN  DELETE FROM C##Vasilisa.Addresses;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('No addresses found.');  ROLLBACK;  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('All addresses deleted successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleting addresses: ' || SQLERRM);  RAISE;  END;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error deleting addresses: ' || SQLERRM);  RAISE;  END;  BEGIN  DeleteAllAddresses;  END;  -------------- Вывод всех адресов --------------  SELECT \* FROM C##Vasilisa.Addresses;  -------------- Добавление дома --------------  CREATE OR REPLACE PROCEDURE InsertHouse(  p\_HouseName IN NVARCHAR2,  p\_NumberOfFlats IN NUMBER,  p\_NumberOfPorches IN NUMBER,  p\_AddressId IN NUMBER,  p\_UserId IN NUMBER  )  IS  BEGIN  INSERT INTO C##Vasilisa.Houses (HouseName, NumberOfFlats, NumberOfPorches, AddressId, UserId)  VALUES (p\_HouseName, p\_NumberOfFlats, p\_NumberOfPorches, p\_AddressId, p\_UserId);    COMMIT;    DBMS\_OUTPUT.PUT\_LINE('House inserted successfully.');  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error inserting house: No data found.');  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error inserting house: ' || SQLERRM);  END;  BEGIN  C##Vasilisa.InsertHouse('Sample House', 10, 2, 1, 1);  END;  -------------- Изменение данных дома --------------  CREATE OR REPLACE PROCEDURE AlterHouse(  p\_HouseId IN NUMBER,  p\_HouseName IN NVARCHAR2,  p\_NumberOfFlats IN NUMBER,  p\_NumberOfPorches IN NUMBER,  p\_AddressId IN NUMBER,  p\_UserId IN NUMBER  )  IS  BEGIN  UPDATE C##Vasilisa.Houses  SET HouseName = p\_HouseName,  NumberOfFlats = p\_NumberOfFlats,  NumberOfPorches = p\_NumberOfPorches,  AddressId = p\_AddressId,  UserId = p\_UserId  WHERE HouseId = p\_HouseId;    IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('Error altering house: House with HouseId ' || p\_HouseId || ' not found.');  ROLLBACK;  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('House altered successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error altering house: ' || SQLERRM);  END;  BEGIN  C##Vasilisa.AlterHouse(6,'Altered House', 10, 2, 2, 3);  END;  -------------- Удаление дома --------------  CREATE OR REPLACE PROCEDURE DeleteHouse(  p\_HouseId IN NUMBER  )  IS  BEGIN  DELETE FROM C##Vasilisa.Houses  WHERE HouseId = p\_HouseId;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('House with ID ' || p\_HouseId || ' not found.');  ROLLBACK;  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('House deleted successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleting house: ' || SQLERRM);  END;  BEGIN  C##Vasilisa.DeleteHouse(1);  END;  -------------- Удаление всех домов --------------  CREATE OR REPLACE PROCEDURE DeleteAllHouses  IS  BEGIN  DELETE FROM C##Vasilisa.Houses;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('No houses found.');  ROLLBACK;  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('All houses deleted successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleting houses: ' || SQLERRM);  END;  BEGIN  DeleteAllHouses;  END;  -------------- Вывод всех домов --------------  SELECT \* FROM C##Vasilisa.Houses;  -------------- Добавление подъезда --------------  CREATE OR REPLACE PROCEDURE InsertPorche (  p\_PorchNumber IN Porches.PorchNumber%TYPE,  p\_HouseId IN Porches.HouseId%TYPE  ) AS  BEGIN  BEGIN  INSERT INTO C##Vasilisa.Porches (PorchNumber, HouseId)  VALUES (p\_PorchNumber, p\_HouseId);    COMMIT;  EXCEPTION  WHEN INVALID\_NUMBER THEN  DBMS\_OUTPUT.PUT\_LINE('Invalid HouseId provided: ' || p\_HouseId);  ROLLBACK;  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  ROLLBACK;  END;  END;  BEGIN  C##Vasilisa.InsertPorche(1, 1);  END;  -------------- Добавление подъездов и квартир по id дома --------------  CREATE OR REPLACE PROCEDURE InsertPorchesByHouseId (  p\_HouseId IN Porches.HouseId%TYPE  ) AS  v\_Amount NUMBER;  v\_NumberOfPorches NUMBER;  v\_PorchId Porches.PorchId%TYPE;  BEGIN  SELECT NumberOfFlats / NumberOfPorches INTO v\_Amount  FROM Houses  WHERE HouseId = p\_HouseId;  SELECT NumberOfPorches INTO v\_NumberOfPorches  FROM Houses  WHERE HouseId = p\_HouseId;  FOR i IN 1..v\_NumberOfPorches LOOP  BEGIN  INSERT INTO Porches (PorchNumber, HouseId)  VALUES (i, p\_HouseId)  RETURNING PorchId INTO v\_PorchId;  FOR j IN 1..v\_Amount LOOP  INSERT INTO Flats (FlatNumber, PorchId)  VALUES (((i - 1) \* v\_Amount) + j, v\_PorchId);  END LOOP;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Porch ' || i || ' created successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error creating porch: ' || SQLERRM);  RAISE;  END;  END LOOP;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('House with ID ' || p\_HouseId || ' not found.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  END;  BEGIN  InsertPorchesByHouseId(2);  END;  -------------- Изменение данных подъезда --------------  CREATE OR REPLACE PROCEDURE AlterPorch(  p\_PorchId IN INTEGER,  p\_PorchNumber IN INTEGER,  p\_HouseId IN INTEGER  )  IS  BEGIN  UPDATE C##Vasilisa.Porches  SET PorchNumber = p\_PorchNumber,  HouseId = p\_HouseId  WHERE PorchId = p\_PorchId;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('Porche with ID ' || p\_PorchId || ' not found.');  ROLLBACK;  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Porche altered successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error altering porche: ' || SQLERRM);  RAISE;  END;  BEGIN  C##Vasilisa.AlterPorch(1, 10, 1);  END;  -------------- Удаление подъезда --------------  CREATE OR REPLACE PROCEDURE DeletePorch(  p\_PorchId IN INTEGER  )  IS  BEGIN  DELETE FROM C##Vasilisa.Porches  WHERE PorchId = p\_PorchId;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('Porch with ID ' || p\_PorchId || ' not found.');  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Porch deleted successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleting porch: ' || SQLERRM);  RAISE;  END;  BEGIN  C##Vasilisa.DeletePorch(1);  END;  -------------- Удаление всех подъездов --------------  CREATE OR REPLACE PROCEDURE DeleteAllPorches  IS  BEGIN  BEGIN  DELETE FROM C##Vasilisa.Porches;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('No porches found.');  ROLLBACK;  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('All porches deleted successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleting porches: ' || SQLERRM);  RAISE;  END;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error deleting addresses: ' || SQLERRM);  RAISE;  END;  BEGIN  DeleteAllPorches();  END;  -------------- Вывод всех подъездов --------------  SELECT \* FROM C##Vasilisa.Porches;  -------------- Добавление квартиры --------------  CREATE OR REPLACE PROCEDURE InsertFlat(  p\_FlatNumber IN NUMBER,  p\_PorchId IN NUMBER  )  IS  BEGIN  INSERT INTO Flats (FlatNumber, PorchId)  VALUES (p\_FlatNumber, p\_PorchId);  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Flat inserted successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error inserting flat: ' || SQLERRM);  RAISE;  END;  BEGIN  C##Vasilisa.InsertFlat(1, 4);  END;  -------------- Изменение квартиры --------------  CREATE OR REPLACE PROCEDURE AlterFlat(  p\_FlatId IN NUMBER,  p\_FlatNumber IN NUMBER,  p\_PorchId IN NUMBER  )  IS  BEGIN  UPDATE Flats  SET FlatNumber = p\_FlatNumber,  PorchId = p\_PorchId  WHERE FlatId = p\_FlatId;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('Flat with ID ' || p\_FlatId || ' not found.');  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Flat altered successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error altering flat: ' || SQLERRM);  RAISE;  END;  BEGIN  C##Vasilisa.AlterFlat(1, 1, 1);  END;  -------------- Удаление квартиры --------------  CREATE OR REPLACE PROCEDURE DeleteFlat(  p\_FlatId IN NUMBER  )  IS  BEGIN  DELETE FROM Flats  WHERE FlatId = p\_FlatId;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('Flat with ID ' || p\_FlatId || ' not found.');  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Flat deleted successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleting flat: ' || SQLERRM);  RAISE;  END;  BEGIN  C##Vasilisa.AlterFlat(1);  END;  -------------- Удаление всех квартир --------------  CREATE OR REPLACE PROCEDURE DeleteAllFlats  IS  BEGIN  BEGIN  DELETE FROM C##Vasilisa.Flats;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('No flats found.');  ROLLBACK;  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('All flats deleted successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleting flats: ' || SQLERRM);  RAISE;  END;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error deleting flats: ' || SQLERRM);  RAISE;  END;  BEGIN  DeleteAllFlats();  END;  -------------- Вывод всех квартир --------------  SELECT \* FROM C##Vasilisa.Flats ORDER BY FLATNUMBER;  -------------- Добавление жильца собственника --------------  CREATE OR REPLACE PROCEDURE InsertOwner(  p\_Surname IN NVARCHAR2,  p\_Name IN NVARCHAR2,  p\_Patronymic IN NVARCHAR2,  p\_AdditionalInfo IN NVARCHAR2,  p\_CurrentDebt IN NUMBER,  p\_PhoneNumberId IN NUMBER,  p\_OwnerStatusId IN NUMBER,  p\_FlatId IN NUMBER  )  IS  BEGIN  INSERT INTO C##Vasilisa.Owners (Surname, Name, Patronymic, AdditionalInfo, CurrentDebt, PhoneNumberId, OwnerStatusId, FlatId)  VALUES (p\_Surname, p\_Name, p\_Patronymic, p\_AdditionalInfo, p\_CurrentDebt, p\_PhoneNumberId, p\_OwnerStatusId, p\_FlatId);  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Owner inserted successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error inserting owner: ' || SQLERRM);  RAISE;  END;  BEGIN  C##Vasilisa.InsertOwner('Smith', 'John', 'Robertovich', 'Additional info', 5000, 1, 1, 4);  END;  -------------- Изменение данных жильца собственника --------------  CREATE OR REPLACE PROCEDURE AlterOwner(  p\_OwnerId IN NUMBER,  p\_Surname IN NVARCHAR2,  p\_Name IN NVARCHAR2,  p\_Patronymic IN NVARCHAR2,  p\_AdditionalInfo IN NVARCHAR2,  p\_CurrentDebt IN NUMBER,  p\_PhoneNumberId IN NUMBER,  p\_OwnerStatusId IN NUMBER,  p\_FlatId IN NUMBER  )  IS  BEGIN  UPDATE C##Vasilisa.Owners  SET Surname = p\_Surname,  Name = p\_Name,  Patronymic = p\_Patronymic,  AdditionalInfo = p\_AdditionalInfo,  CurrentDebt = p\_CurrentDebt,  PhoneNumberId = p\_PhoneNumberId,  OwnerStatusId = p\_OwnerStatusId,  FlatId = p\_FlatId  WHERE OwnerId = p\_OwnerId;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('Owner with ID ' || p\_OwnerId || ' not found.');  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Owner altered successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error altering owner: ' || SQLERRM);  RAISE;  END;  BEGIN  C##Vasilisa.AlterOwner(2,'Smith', 'John', 'Robertovich', 'Additional info', 5000, 1, 1, 1);  END;  -------------- Удаление жильца собственника --------------  CREATE OR REPLACE PROCEDURE DeleteOwner(  p\_OwnerId IN NUMBER  )  IS  BEGIN  DELETE FROM C##Vasilisa.Owners  WHERE OwnerId = p\_OwnerId;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('Owner with ID ' || p\_OwnerId || ' not found.');  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Owner deleted successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleting owner: ' || SQLERRM);  RAISE;  END;  BEGIN  C##Vasilisa.DeleteOwner(2);  END;  -------------- Удаление всех жильцов собственников --------------  CREATE OR REPLACE PROCEDURE DeleteAllOwners  IS  BEGIN  BEGIN  DELETE FROM C##Vasilisa.Owners;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('No owners found.');  ROLLBACK;  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('All owners deleted successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleting owners: ' || SQLERRM);  RAISE;  END;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error deleting owners: ' || SQLERRM);  RAISE;  END;  BEGIN  DeleteAllOwners();  END;  -------------- Найти жильцов собственников по фамилии --------------  CREATE OR REPLACE PROCEDURE FindOwnerBySurname(  p\_Surname IN NVARCHAR2  )  IS  v\_OwnerId Owners.OwnerId%TYPE;  v\_Name Owners.Name%TYPE;  v\_Patronymic Owners.Patronymic%TYPE;  v\_AdditionalInfo Owners.AdditionalInfo%TYPE;  v\_CurrentDebt Owners.CurrentDebt%TYPE;  v\_PhoneNumberId Owners.PhoneNumberId%TYPE;  v\_OwnerStatusId Owners.OwnerStatusId%TYPE;  v\_FlatId Owners.FlatId%TYPE;    CURSOR c\_Owners IS  SELECT OwnerId, Name, Patronymic, AdditionalInfo, CurrentDebt, PhoneNumberId, OwnerStatusId, FlatId  FROM C##Vasilisa.Owners  WHERE Surname = p\_Surname;  BEGIN  BEGIN  OPEN c\_Owners;  LOOP  FETCH c\_Owners INTO v\_OwnerId, v\_Name, v\_Patronymic, v\_AdditionalInfo, v\_CurrentDebt, v\_PhoneNumberId, v\_OwnerStatusId, v\_FlatId;  EXIT WHEN c\_Owners%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE('-------------------------');  DBMS\_OUTPUT.PUT\_LINE('Owner ID: ' || v\_OwnerId);  DBMS\_OUTPUT.PUT\_LINE('Name: ' || v\_Name);  DBMS\_OUTPUT.PUT\_LINE('Patronymic: ' || v\_Patronymic);  DBMS\_OUTPUT.PUT\_LINE('Additional Info: ' || v\_AdditionalInfo);  DBMS\_OUTPUT.PUT\_LINE('Current Debt: ' || v\_CurrentDebt);  DBMS\_OUTPUT.PUT\_LINE('Phone Number ID: ' || v\_PhoneNumberId);  DBMS\_OUTPUT.PUT\_LINE('Owner Status ID: ' || v\_OwnerStatusId);  DBMS\_OUTPUT.PUT\_LINE('Flat ID: ' || v\_FlatId);  DBMS\_OUTPUT.PUT\_LINE('-------------------------');  END LOOP;  CLOSE c\_Owners;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('No owners found with surname ' || p\_Surname);  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error finding owners: ' || SQLERRM);  END;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error finding owners: ' || SQLERRM);  END;  BEGIN  C##Vasilisa.FindOwnerBySurname('Smith');  END;  -------------- Найти жильцов собственников номеру квартиры и названию дома --------------  CREATE OR REPLACE PROCEDURE FindOwnerByFlatNumber(  p\_FlatNumber IN NUMBER,  p\_HouseName IN NVARCHAR2  )  IS  v\_OwnerName NVARCHAR2(50);  BEGIN  SELECT O.Surname || ', ' || O.Name AS OwnerName  INTO v\_OwnerName  FROM C##Vasilisa.Owners O  JOIN C##Vasilisa.Flats F ON O.FlatId = F.FlatId  JOIN C##Vasilisa.Porches P ON F.PorchId = P.PorchId  JOIN C##Vasilisa.Houses H ON P.HouseId = H.HouseId  WHERE F.FlatNumber = p\_FlatNumber  AND H.HouseName = p\_HouseName;  IF v\_OwnerName IS NULL THEN  DBMS\_OUTPUT.PUT\_LINE('Owner not found.');  ELSE  DBMS\_OUTPUT.PUT\_LINE('Owner: ' || v\_OwnerName);  END IF;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('Owner not found.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error finding owner: ' || SQLERRM);  END;  BEGIN  C##Vasilisa.FindOwnerByFlatNumber(1, 'Sample House');  END;  -------------- Вывод всех жильцов собственников --------------  SELECT \* FROM C##Vasilisa.Owners;  -------------- Добавление номера --------------  CREATE OR REPLACE PROCEDURE InsertPhoneNumber(  p\_MobilePhone IN NVARCHAR2,  p\_HomePhone IN NVARCHAR2,  p\_AdditionalPhone IN NVARCHAR2  )  IS  BEGIN  INSERT INTO C##Vasilisa.PhoneNumbers (MobilePhone, HomePhone, AdditionalPhone)  VALUES (p\_MobilePhone, p\_HomePhone, p\_AdditionalPhone);    COMMIT;    DBMS\_OUTPUT.PUT\_LINE('Phone number inserted successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error inserting phone number: ' || SQLERRM);  RAISE;  END;  BEGIN  C##Vasilisa.InsertPhoneNumber('+375123456789', '123456', '+375987654321');  END;  -------------- Изменение номера --------------  CREATE OR REPLACE PROCEDURE AlterPhoneNumber(  p\_PhoneNumberId IN NUMBER,  p\_MobilePhone IN NVARCHAR2,  p\_HomePhone IN NVARCHAR2,  p\_AdditionalPhone IN NVARCHAR2  )  IS  BEGIN  UPDATE C##Vasilisa.PhoneNumbers  SET MobilePhone = p\_MobilePhone,  HomePhone = p\_HomePhone,  AdditionalPhone = p\_AdditionalPhone  WHERE PhoneNumberId = p\_PhoneNumberId;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('Phone number with ID ' || p\_PhoneNumberId || ' not found.');  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Phone number altered successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error altering phone number: ' || SQLERRM);  RAISE;  END;  BEGIN  C##Vasilisa.AlterPhoneNumber(2, '+375123456789', '123456', '+375987654321');  END;  -------------- Удаление номера --------------  CREATE OR REPLACE PROCEDURE DeletePhoneNumber(  p\_PhoneNumberId IN NUMBER  )  IS  BEGIN  DELETE FROM C##Vasilisa.PhoneNumbers  WHERE PhoneNumberId = p\_PhoneNumberId;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('Phone number with ID ' || p\_PhoneNumberId || ' not found.');  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Phone number deleted successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleting phone number: ' || SQLERRM);  RAISE;  END;  BEGIN  C##Vasilisa.DeletePhoneNumber(1);  END;  -------------- Удаление всех номеров --------------  CREATE OR REPLACE PROCEDURE DeleteAllPhoneNumbers  IS  BEGIN  BEGIN  DELETE FROM C##Vasilisa.PhoneNumbers;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('No phone numbers found.');  ROLLBACK;  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('All phone numbers deleted successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleting phone numbers: ' || SQLERRM);  RAISE;  END;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error deleting phone numbers: ' || SQLERRM);  RAISE;  END;  -------------- Добавление контакта --------------  CREATE OR REPLACE PROCEDURE InsertContact(  p\_Surname IN NVARCHAR2,  p\_Name IN NVARCHAR2,  p\_Patronymic IN NVARCHAR2,  p\_Position IN NVARCHAR2,  p\_PhoneNumberId IN NUMBER,  p\_UserId IN NUMBER  )  IS  BEGIN  INSERT INTO C##Vasilisa.Contacts (Surname, Name, Patronymic, Position, PhoneNumberId, UserId)  VALUES (p\_Surname, p\_Name, p\_Patronymic, p\_Position, p\_PhoneNumberId, p\_UserId);  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Contact inserted successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error inserting contact: ' || SQLERRM);  RAISE;  END;  BEGIN  C##Vasilisa.InsertContact('Smith', 'John', 'Doe', 'Manager', 1, 1);  END;  -------------- Изменение контакта --------------  CREATE OR REPLACE PROCEDURE AlterContact(  p\_ContactId IN NUMBER,  p\_Surname IN NVARCHAR2,  p\_Name IN NVARCHAR2,  p\_Patronymic IN NVARCHAR2,  p\_Position IN NVARCHAR2,  p\_PhoneNumberId IN NUMBER,  p\_UserId IN NUMBER  )  IS  BEGIN  UPDATE C##Vasilisa.Contacts  SET Surname = p\_Surname,  Name = p\_Name,  Patronymic = p\_Patronymic,  Position = p\_Position,  PhoneNumberId = p\_PhoneNumberId,  UserId = p\_UserId  WHERE ContactId = p\_ContactId;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('Contact with ID ' || p\_ContactId || ' not found.');  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Contact altered successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error altering contact: ' || SQLERRM);  RAISE;  END;  BEGIN  C##Vasilisa.AlterContact(1, 'Smith', 'John', 'Doe', 'Manager', 1, 1);  END;  -------------- Удаление контакта --------------  CREATE OR REPLACE PROCEDURE DeleteContact(  p\_ContactId IN NUMBER  )  IS  BEGIN  DELETE FROM C##Vasilisa.Contacts  WHERE ContactId = p\_ContactId;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('Contact with ID ' || p\_ContactId || ' not found.');  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Contact deleted successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleting contact: ' || SQLERRM);  RAISE;  END;  BEGIN  C##Vasilisa.DeleteContact(2);  END;  -------------- Удаление всех контактов --------------  CREATE OR REPLACE PROCEDURE DeleteAllPhoneNumbers  IS  BEGIN  BEGIN  DELETE FROM C##Vasilisa.PhoneNumbers;  IF SQL%ROWCOUNT = 0 THEN  DBMS\_OUTPUT.PUT\_LINE('No phone numbers found.');  ROLLBACK;  ELSE  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('All phone numbers deleted successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleting phone numbers: ' || SQLERRM);  RAISE;  END;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error deleting phone numbers: ' || SQLERRM);  RAISE;  END;  BEGIN  DeleteAllPhoneNumbers();  END;  -------------- Вывод всех контактов --------------  SELECT \* FROM C##Vasilisa.Contacts; |

# ПРИЛОЖЕНИЕ Г

Скрипт создания индексов

|  |
| --- |
| CREATE INDEX idx\_AddressCountry ON C##Vasilisa.Addresses(Country);  CREATE INDEX idx\_HousesHouseName ON C##Vasilisa.Houses(HouseName);  CREATE INDEX idx\_OwnersSurname ON C##Vasilisa.Owners(Surname); |

# ПРИЛОЖЕНИЕ Д

Скрипт процедур экспорта и импорта данных в XML-формат

|  |
| --- |
| -------------- Процедура экспорта данных в XML --------------  CREATE OR REPLACE PROCEDURE ExportHousesToXML  IS  v\_xml XMLType;  v\_clob CLOB;  v\_file UTL\_FILE.FILE\_TYPE;  BEGIN  -- Выборка данных из таблицы Houses в XMLType  SELECT XMLElement(  "Houses",  XMLAgg(  XMLElement(  "House",  XMLForest(  HouseId AS "HouseId",  HouseName AS "HouseName",  NumberOfFlats AS "NumberOfFlats",  NumberOfPorches AS "NumberOfPorches",  AddressId AS "AddressId",  UserId AS "UserId"  )  )  )  )  INTO v\_xml  FROM Houses;  -- Преобразование XMLType в CLOB  v\_clob := v\_xml.getClobVal();  -- Создание и запись XML в файл  v\_file := UTL\_FILE.FOPEN('EXPORT\_DIR', 'houses.xml', 'W');  UTL\_FILE.PUT\_RAW(v\_file, UTL\_RAW.CAST\_TO\_RAW(v\_clob));  UTL\_FILE.FCLOSE(v\_file);    DBMS\_OUTPUT.PUT\_LINE('Export completed successfully.');  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error occurred while exporting houses to XML.');  DBMS\_OUTPUT.PUT\_LINE(SQLERRM);  END;  -------------- Процедура импорта данных из XML --------------  CREATE OR REPLACE PROCEDURE ImportHousesFromXML  IS  v\_xml XMLType;  v\_house\_name NVARCHAR2(20);  v\_number\_of\_flats NUMBER;  v\_number\_of\_porches NUMBER;  v\_address\_id NUMBER;  v\_user\_id NUMBER;  BEGIN  -- Чтение XML из файла  SELECT XMLType(  UTL\_RAW.CAST\_TO\_VARCHAR2(UTL\_FILE.READ\_RAW('IMPORT\_DIR', 'houses.xml', UTL\_FILE.GET\_FILE\_SIZE('IMPORT\_DIR', 'houses.xml')))  )  INTO v\_xml  FROM dual;  -- Импорт данных из XML в таблицу  FOR r IN (  SELECT ExtractValue(value(h), '/House/HouseName') AS house\_name,  ExtractValue(value(h), '/House/NumberOfFlats') AS number\_of\_flats,  ExtractValue(value(h), '/House/NumberOfPorches') AS number\_of\_porches,  ExtractValue(value(h), '/House/AddressId') AS address\_id,  ExtractValue(value(h), '/House/UserId') AS user\_id  FROM TABLE(XMLSequence(v\_xml.extract('/Houses/House'))) h  )  LOOP  v\_house\_name := r.house\_name;  v\_number\_of\_flats := TO\_NUMBER(r.number\_of\_flats);  v\_number\_of\_porches := TO\_NUMBER(r.number\_of\_porches);  v\_address\_id := TO\_NUMBER(r.address\_id);  v\_user\_id := TO\_NUMBER(r.user\_id);    -- Вставка данных в таблицу Houses  INSERT INTO Houses (HouseName, NumberOfFlats, NumberOfPorches, AddressId, UserId)  VALUES (v\_house\_name, v\_number\_of\_flats, v\_number\_of\_porches, v\_address\_id, v\_user\_id);  END LOOP;    DBMS\_OUTPUT.PUT\_LINE('Import completed successfully.');  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error occurred while importing houses from XML.');  DBMS\_OUTPUT.PUT\_LINE(SQLERRM);  END; |

# ПРИЛОЖЕНИЕ Е

Скрипт создания блока для тестирования производительности

|  |
| --- |
| -- включаем вывод данных в output  set serveroutput on;  declare  v\_t1 number; -- время начала  v\_t2 number; -- время завершения  v\_cpu1 number; -- время ЦПУ до  v\_cpu2 number; -- время ЦПУ после  v\_t\_res number; -- общее время выполнения  v\_cpu\_res number; -- общее время выполнения CPU  begin  -- помещаем общее время и время ЦПУ в переменные  select t.hsecs  ,dbms\_utility.get\_cpu\_time  into v\_t1  ,v\_cpu1  from v$timer t;    -- запустим 100000 раз цикл, выполняющий процедуру вставки в таблицу Addresses  begin  for i IN 1..100000  loop  C##Vasilisa.InsertAddress('TEST', 'TEST', 'TEST', 'TEST', 123, 'A');  end loop;  end;    -- снова помещаем общее время и время ЦПУ в переменные  select t.hsecs  ,dbms\_utility.get\_cpu\_time  into v\_t2  ,v\_cpu2  from v$timer t;    -- считаем общее время выполнения в сотых долях секунды  v\_t\_res := v\_t2 - v\_t1;    -- считаем общее время ЦПУ в сотых долях секунды  v\_cpu\_res := v\_cpu2 - v\_cpu1;    -- вывод результата на экран  dbms\_output.put\_line('Общее время выполнения в секундах: '||to\_char(v\_t\_res/100,'0.00'));  dbms\_output.put\_line('Общее время ЦПУ в секундах: '||to\_char(v\_cpu\_res/100,'0.00'));  -- откат изменения  rollback;  end;  CREATE INDEX idx\_InsertAddress ON C##Vasilisa.Addresses(Country); |