

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 \_\_\_\_\_

## Содержание

Введение .....	5
1 Постановка задачи .....	7
1.1 Аналитический обзор аналогов и литературных источников.....	7
1.2 Анализ и разработка функциональных требований.....	10
2 Разработка модели базы данных .....	12
3 Разработка необходимых объектов.....	14
3.1 Таблицы .....	14
3.2 Пользователи .....	16
3.3 Хранимые процедуры.....	17
3.4 Индексы .....	17
4 Описание процедур экспорта и импорта .....	18
5 Тестирование производительности.....	19
6 Описание технологии резервного копирования и восстановления и ее применения в базе данных.....	21
7 Руководство пользователя .....	24
Заключение .....	32
Список используемых источников.....	33
ПРИЛОЖЕНИЕ А .....	34
ПРИЛОЖЕНИЕ Б .....	37
ПРИЛОЖЕНИЕ В .....	39
ПРИЛОЖЕНИЕ Г .....	62
ПРИЛОЖЕНИЕ Д .....	63
ПРИЛОЖЕНИЕ Е .....	65

## 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 общего 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 C-подобным синтаксисом – C#, 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.

Многу 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.

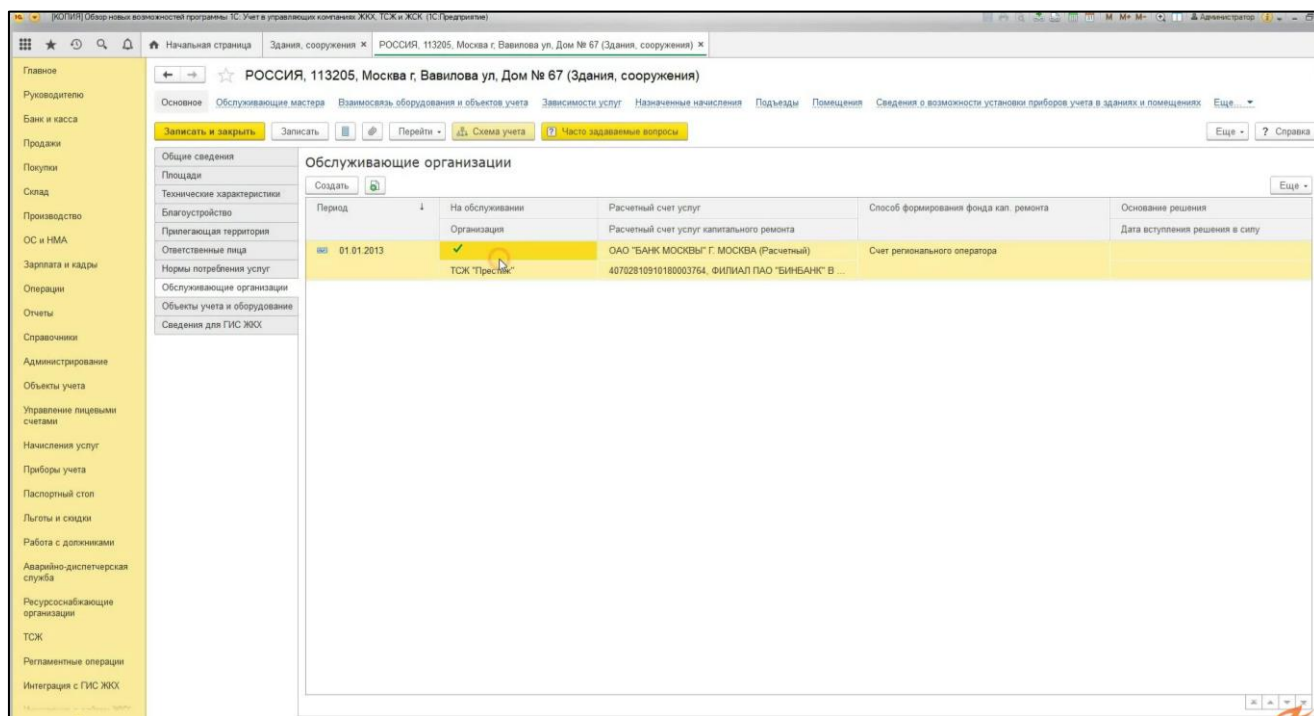


Рисунок 1.1 – Интерфейс программного средства «Председатель ТСЖ»

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.

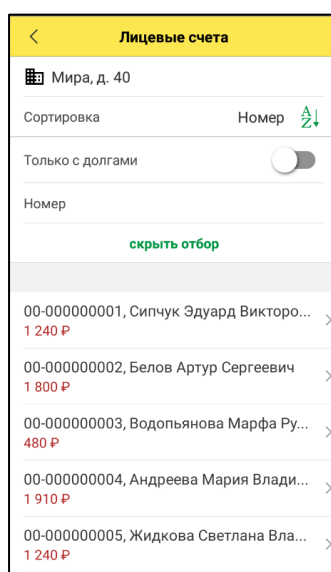


Рисунок 1.2 – Интерфейс «Председатель 365»

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

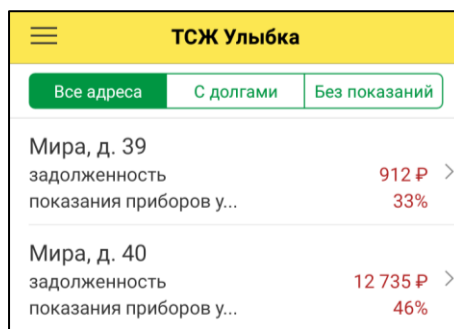


Рисунок 1.3 – Интерфейс «Председатель 365»

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.

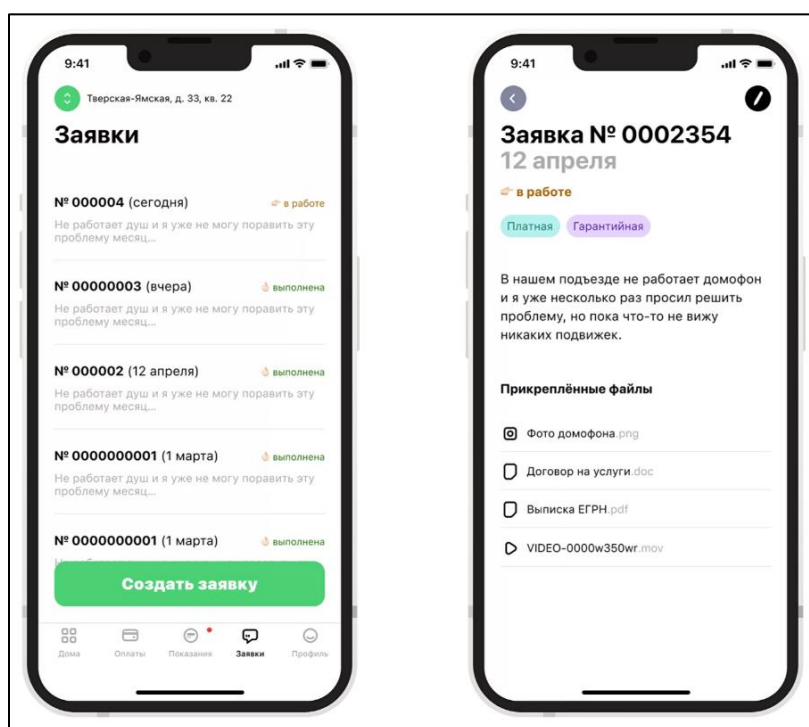


Рисунок 1.4 – Интерфейс «Домá»

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 с 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 сформированным 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 упростить 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.

### 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.

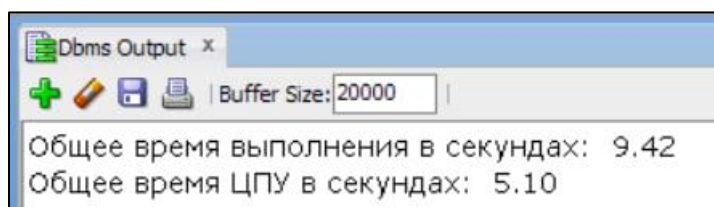


Рисунок 5.1 – Затраченное время на выполнение анонимного блока без индекса

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.

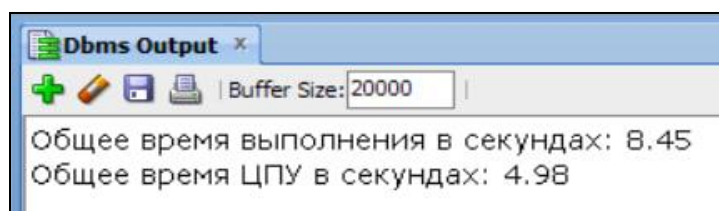


Рисунок 5.2 – Затраченное время на выполнение анонимного блока с индексом

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.

Кроме того, в Oracle 12c также доступны другие инструменты и технологии для резервного копирования и восстановления данных, такие как Data Pump, Export/Import, flashback и т.д. Они также могут использоваться для создания и восстановления резервных копий.

Применение резервного копирования и восстановления в базе данных Oracle 12c обеспечивает надежность и сохранность данных, позволяет избежать потери данных при сбоях и обеспечивает возможность восстановления данных в их последнем состоянии.

Также, это особенно важно для критически важных приложений и систем, в которых необходимо обеспечить непрерывность работы и сохранность данных в любых условиях.

В базе данных Oracle 12c можно выполнить резервное копирование с использованием инструмента управления резервным копированием (Recovery Manager, RMAN). RMAN предоставляет мощные средства для создания и управления резервными копиями базы данных.

Для начала осуществляется подготовка конфигурации RMAN: создается файл конфигурации RMAN (обычно с расширением ".rman") для указания параметров резервного копирования, таких как место хранения копий данных и логов, типы резервных копий и так далее.

Затем осуществляется подключение к базе данных: запуск RMAN и подключение к базе данных с помощью команды "CONNECT" с указанием учетных данных администратора базы данных (рисунок 6.1).



```

D:\app\OracleUser\product\12.1.0\dbhome_1\BIN\rman.exe
Recovery Manager: Release 12.1.0.1.0 - Production on Thu May 11 21:23:11 2023
Copyright (c) 1982, 2013, Oracle and/or its affiliates. All rights reserved.
RMAN> CONNECT TARGET C##Vasilisa/Pa$$w0rd@orcl1
connected to target database: ORCL1 (DBID=1499091797)
RMAN>
  
```

Рисунок 6.1 – Подключение к базе данных в RMAN

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

Наконец, выполняется резервное копирование: используются команды RMAN для запуска операции резервного копирования. Например, для полного копирования базы данных можно выполнить команду "BACKUP DATABASE" (рисунок 6.2).

```
RMAN> BACKUP DATABASE tag "full_database_datafiles";
```

Рисунок 6.2 – Команда резервного копирования всех файлов базы данных

На рисунке 6.3 изображен процесс создания резервного копирования.

```

using target database control file instead of recovery catalog
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=50 device type=DISK
channel ORA_DISK_1: starting full datafile backup set
channel ORA_DISK_1: specifying datafile(s) in backup set
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:07
channel ORA_DISK_1: starting full datafile backup set
channel ORA_DISK_1: specifying datafile(s) in backup set
input datafile file number=000004 name=C:\APP\ORA_INSTALL_USER\ORADATA\ORCL\PDBSE
ED\SYSAUX01.DBF
input datafile file number=000002 name=C:\APP\ORA_INSTALL_USER\ORADATA\ORCL\PDBSE
ED\SYSTEM01.DBF
  
```

Рисунок 6.3 – Команда резервного копирования всех файлов базы данных

RMAN предоставляет возможность управления созданными резервными копиями. Вы можете проверить их статус, восстановить данные из копий или удалить устаревшие копии.

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

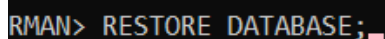
Для восстановления данных из резервной копии в базе данных Oracle 12c можно использовать инструмент управления резервным копированием (Recovery Manager, RMAN). RMAN предоставляет функциональность для восстановления данных на основе созданных резервных копий.

Сначала осуществляется подготовка конфигурации RMAN: проверка, существует ли файл конфигурации RMAN (.rman), который указывает параметры восстановления, такие как место хранения резервных копий и логов.

Затем 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.

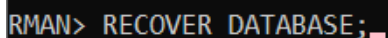
Наконец, происходит выполнение операции восстановления: использование команды RMAN для запуска операции восстановления. Например, для восстановления полной базы данных из резервной копии можно выполнить команду "RESTORE DATABASE" (рисунок 6.4).



```
RMAN> RESTORE DATABASE;_
```

Рисунок 6.4 – Команда восстановления всех файлов базы данных

Затем выполняется команда "RECOVER DATABASE" для применения журналов восстановления (рисунок 6.5).



```
RMAN> RECOVER DATABASE;_
```

Рисунок 6.5 – Команда применения журналов восстановления

По окончании процесса восстановления выводится сообщение о затраченном времени (рисунок 6.6).



```
complete, elapsed time: 00:00:07
```

Рисунок 6.6 – Команда применения журналов восстановления

RMAN предоставляет возможности для управления процессом восстановления. Можно проверить статус восстановления, просмотреть журналы восстановления, управлять параметрами восстановления.

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.

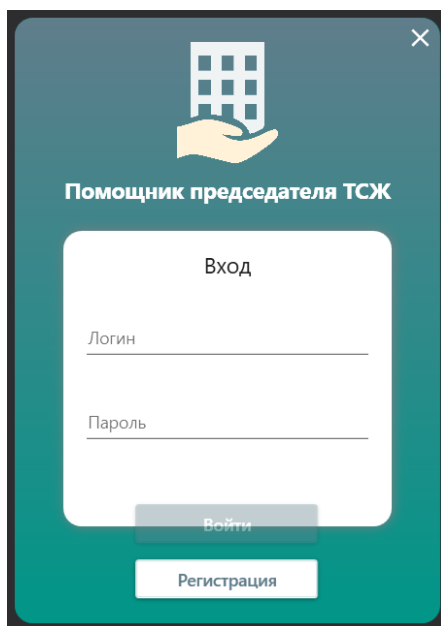


Рисунок 7.1 – Страница входа

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).

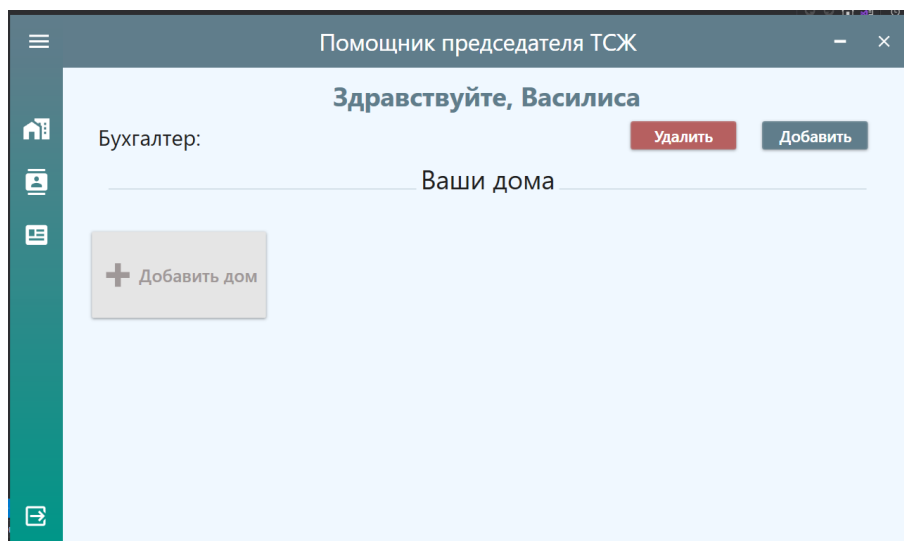


Рисунок 7.2 – Главная страница нового пользователя

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 находиться 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).

№ кв.	Фамилия	Имя	Отчество	Номер телефона	Статус	Долг, р.
1					Не задан	0
2					Не задан	0
3					Не задан	0
4					Не задан	0
5					Не задан	0
6					Не задан	0
7					Не задан	0
8					Не задан	0
9					Не задан	0

Рисунок 7.8 – Активация кнопки «Добавить информацию»

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

№ кв.	Фамилия	Имя	Отчество	Номер телефона	Статус	Долг, р.
1					Не задан	0
2					Не задан	0
3	Кашперко	Елена	Валерьевна	+375446627272	Живет	0
4					Не задан	0
5					Не задан	0
6					Не задан	0
7					Не задан	0
8					Не задан	0
9					Не задан	0

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 жилья, 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).

Фамилия	Имя	Отчество	Должность	Номер телефона
Кашперко	Сергей	Олегович	Заместитель	+375123456789

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).

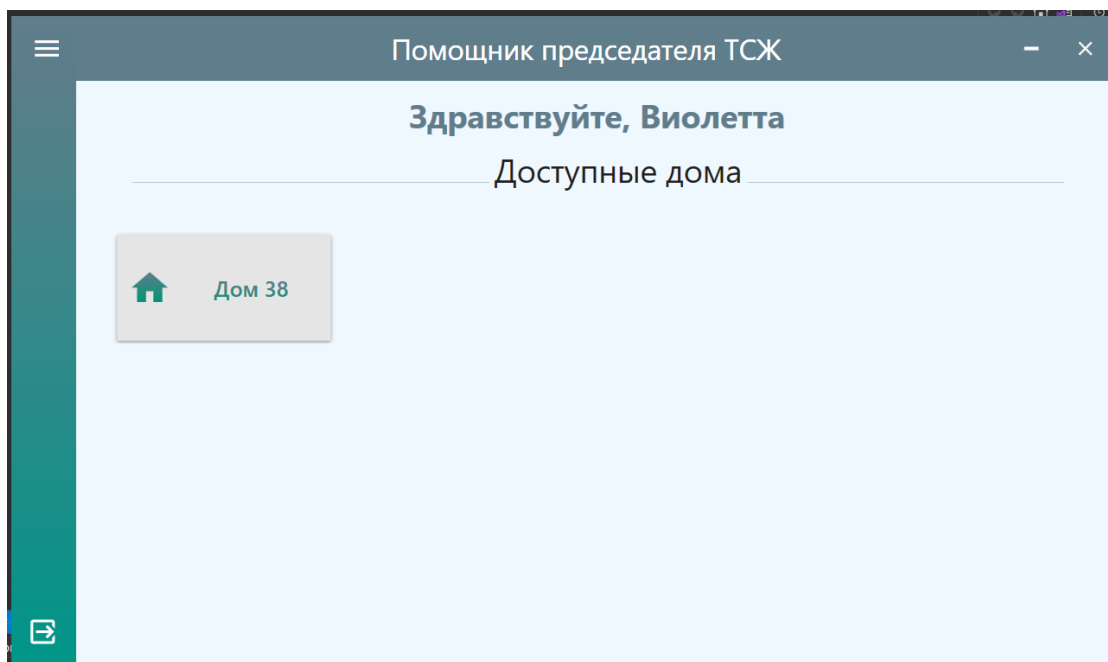


Рисунок 7.15 – Главная страница пользователя с ролью «Бухгалтер»

Таким образом функционал приложения для 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 отличие от председателя (рисунок 7.16).

Помощник председателя ТСЖ

← **Изменение информации о жильце** Дом 38  
Квартира 3

Личные данные	Информация
Фамилия* Кашперко	Мобильный телефон* +375446627272
Имя* Елена	Домашний телефон 123472
Отчество Валерьевна	Доп. мобильный телефон
Дополнительная информация Моя мама	Живет
	Текущая задолженность, р. 0

Сохранить

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 с 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 количества 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 объеме.

### Список 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/xdbs26imp.htm#ADXD3000>. – 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#BRADV89346](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##Yuri;
--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##ACCOUNT-
ANT';
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);
    END LOOP;
END;

```

```

        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;

----- Найдти 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 ad-
dress.');
```

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 ad-
dress. ');
        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, Number-
OfPorches, 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 жилья 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, Additional-
Info, 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 жилья 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 жилья 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, Addition-
alPhone)
    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
    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 num-
ber_of_flats,
            ExtractValue(value(h), '/House/NumberOfPorches') AS num-
ber_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, Ad-
dressId, UserId)
        VALUES (v_house_name, v_number_of_flats, v_number_of_porches, v_ad-
dress_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);
```