МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

«Реализация базы данных аренды автомобилей с применением технологии Spatial and Graph в БД»

Выполнил студент Коршун Никита Игоревич

(Ф.И.О.)

Руководитель работы асс. Нистюк О.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

И.о. зав. кафедрой ст. преп. Блинова Е.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовая работа защищена с оценкой

Минск 2023

**Содержание**

[Введение 3](#_Toc153910337)

[1 Постановка задачи 4](#_Toc153910338)

[1.1 Аналитический обзор аналогов и литературных решений 4](#_Toc153910339)

[1.2 Анализ и разработка функциональных требований 6](#_Toc153910340)

[2 Проектирование базы данных 8](#_Toc153910341)

[3 Разработка необходимых объектов 11](#_Toc153910342)

[3.1 Таблицы 11](#_Toc153910343)

[3.2 Пользователи и роли 13](#_Toc153910344)

[3.3 Процедуры и функции 13](#_Toc153910345)

[4 Описание процедур импорта и экспорта 14](#_Toc153910346)

[4.1 Процедура импорта данных в БД 14](#_Toc153910347)

[4.2 Процедура экспорта данных из БД 14](#_Toc153910348)

[5 Тестирование производительности 15](#_Toc153910349)

[6 Описание технологии применяемой в БД 18](#_Toc153910350)

[6.1 Шифрование 18](#_Toc153910351)

[6.2 Маскирование 19](#_Toc153910352)

[7 Краткое описание приложения 20](#_Toc153910353)

[8 Руководство пользователя 21](#_Toc153910354)

[Заключение 24](#_Toc153910355)

[Список используемых источников 25](#_Toc153910356)

[Приложение А 26](#_Toc153910357)

[Приложение Б 29](#_Toc153910358)

[Приложение В 34](#_Toc153910359)

[Приложение Г 36](#_Toc153910360)



# Введение

Цель данной работы заключается в создании реляционной базы данных для веб-приложения по бронированию авто, которое позволит клиентам брать в аренду транспорт на определенный срок.

База данных – это организованное собрание данных, которое обычно хранится в электронном виде в компьютерной системе. БД используются для хранения, организации и управления большим объемом структурированных и неструктурированных данных. Реляционная база данных является наиболее распространенной формой организации данных, в которой данные представлены в виде таблиц, состоящих из строк и столбцов, где каждый столбец представляет атрибут, а каждая строка представляет кортеж или запись. В данной работе для управления базой данных была выбрана СУБД Oracle, поскольку эта система обладает большим функционалом, что позволяет обеспечить эффективное хранение, обработку и управление данными.

Также необходимо разработать приложение для демонстрации функциональности базы данных и взаимодействия с ней. Приложение было реализовано с использованием фреймворка Django Framework на языке Python и HTML с использованием JavaScript.

Для сохранения данных местоположения автомобилей в курсовой работе применяется технология Spatial. Так же будут использоваться графы для получения связей между элементами.

Основные требования к приложению:

* Реализация ролей администратора, пользователя;
* Добавление и управление автомобилями;
* Резервирование автомобилей с указанием дат начала и окончания аренды;
* Управление заказами и генерация квитанций;
* Анализ данных (статистика, доход, популярные автомобили);
* Управление пользователями.

Содержание данной пояснительной записки отражает этапы выполнения курсового проекта.

1. Постановка задачи

Задачи проекта: разработать архитектуру базы данных с использованием технологий spatial and graph, создать процедуры и функции для работы с базой данных, взаимодействие с которыми будет понятно любому пользователю. Разработать веб-приложение на Python Django, используя встроенные функции Django для работы с базой данных.

Цели приложения состоит в демонстрации возможностей базы данных и предоставление удобного инструмента для бронирования автомобилей. Приложение должно обеспечивать эффективное и надежное бронирование автомобилей, обеспечивая при этом высокую производительность и безопасность данных.

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. Аналитический обзор аналогов и литературных решений

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

На сегодняшний день существует множество аналогичных приложений или веб-сервисов для бронирования автомобилей. Поэтому было рассмотрено несколько аналогов.

Мобильное приложение «CarBooking» – приложение, предназначенное для бронирования автомобилей, а также помогает в выборе автомобиля и предлагает оптимальные решения для проблем, связанных с бронированием.

Интерфейс приведён на рисунке 1.1.

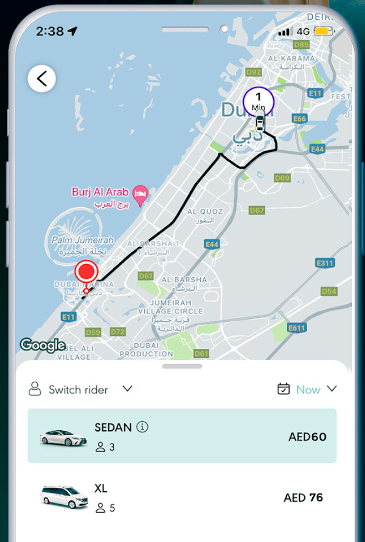


Рисунок 1.1 – Интерфейс приложения «CarBooking»

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

Основные плюсы:

* удобный интерфейс;
* бесплатный;
* имеется чат для общения с оператором;
* имеется карта стоянок.

Основные минусы:

* навязчивая реклама;

Мобильное приложение «CarRental» – приложение является мобильным сервисом для бронирования автомобилей с встроенным калькулятором стоимости.

Интерфейс приведён на рисунке 1.2.

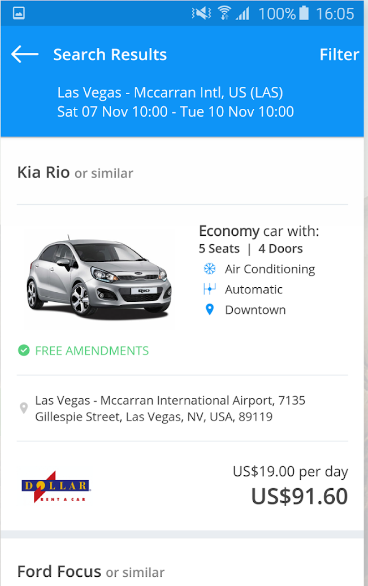


Рисунок 1.2 ­– интерфейс приложения «CarRental»

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

Основные плюсы:

* есть калькулятор стоимости;
* большой выбор автомобилей.

Основные минусы:

* неудобный интерфейс;

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

* 1. 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 ПО.

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

На этом этапе происходит идентификация принципиальных требований методологического и технологического характера, формулируются цели и задачи создаваемого проекта.

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 будут рассматривать общие требования к разрабатываемому курсовому проекту.

К их этим требованиям относятся:

* использование архитектурных шаблонов проектирования;
* использование системы управления базами данных (СУБД);
* доступ к данным должен осуществляться только через соответствующие процедуры.

Следующими требованиями являются требования пользователей приложения, а именно администратора и пользователя.

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 администратора:

* управление базой данных автомобилей;
* управление базой данных бронирований;
* управление базой данных пользователей;
* управление базой данных местоположений.

Возможности пользователя:

* бронирование автомобиля;
* просмотр своих бронирований;
* просмотр доступных автомобилей;
* просмотр местоположений автомобилей

После проведения анализа были выявлены следующие функциональные требования программного средства:

* вся информация должна храниться в базе данных;
* приложение должно производить валидацию вводимых пользователем различных 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.

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.

1. Проектирование 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 из таблиц 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.

Концептуально на данном этапе было выделено 6 сущностей:

* пользователь;
* местоположение;
* автомобиль;
* бронирование;
* узел;
* ребро.

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. В ней показано как будут взаимодействовать таблицы между собой.

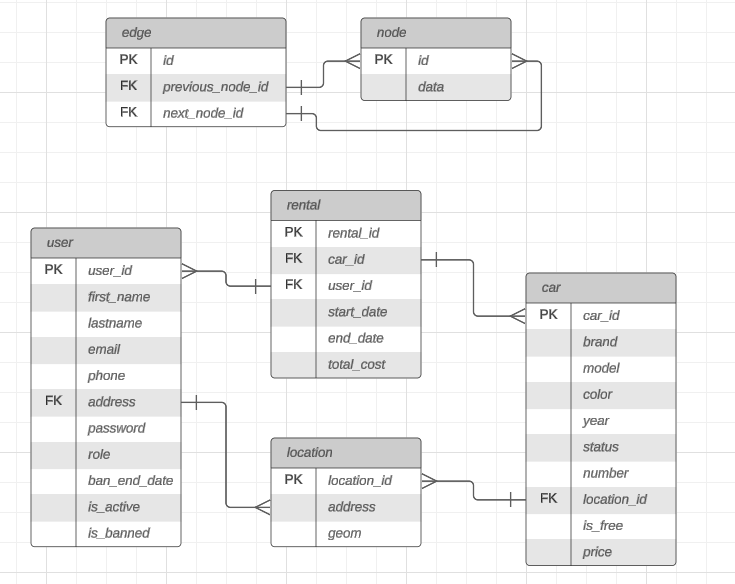


Рисунок 2.1 ­– Логическая модель базы данных

Всего в базе данных содержится 6 таблиц.

Таблица User предназначена для хранения данных зарегистрированных пользователей в базе данный.

Таблица Location предназначена для хранения данных о местоположениях.

Таблица Car предназначена для хранения данных об автомобилях.

Таблица Rental предназначена для хранения данных о бронированиях.

Таблица Node предназначена для хранения данных о узлах.

Таблица Edge предназначена для хранения данных о ребрах.

Так же была разработана use-case диаграмма, предоставленная на рисунке 2.2.

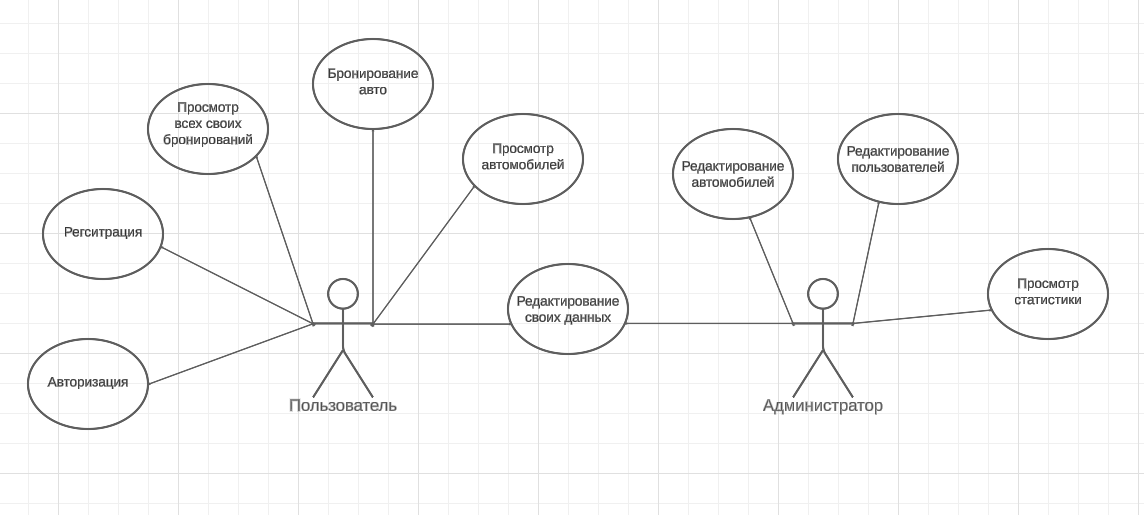


Рисунок 2.2 – Use-case диаграмма

Всего в базе данных 4 пользователя:

Пользователь ADMIN создан для представления администратора в базе данных. Данный пользователь имеет возможность управлять базой данных автомобилей, бронирований и пользователей, просмотра статистики.

Пользователь USER создан для представления пользователя в базе данных. Данный пользователь имеет возможность бронировать автомобиль, просматривать свои бронирования и просматривать доступные автомобили.

1. Разработка необходимых объектов
   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 «Реализация базы данных аренды автомобилей с применением технологии Spatial and Graph в БД» было разработано 6 таблиц: User, Car, Location, Rental, Node, Edge. Они будут реализованы в СУБД Postgresql.

Таблица Users представляет список всех зарегистрированных пользователей в базе данных (Таблица 3.1).

Таблица 3.1 – Столбцы таблицы User

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| user\_id | Идентификатор пользователя | int |
| first\_name | Имя пользователя | varchar |
| last\_name | Фамилия пользователя | varchar |
| email | Почта пользователя | varchar |
| phone | Телефон пользователя | varchar |
| password | Пароль пользователя | varchar |
| role | Дата рождения пользователя | varchar |
| ban\_end\_date | Электронная почта пользователя | date |
| is\_active | Активна ли учетная запись | boolean |
| address\_id | Идентификатор адреса пользователя | int |
| is\_banned | Заблокирована ли учетная запись | boolean |

Таблица Cars представляет список всех автомобилей (таблица 3.2).

Таблица 3.2 – Столбцы таблицы Car

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| car\_id | Идентификатор автомобиля | int |
| brand | Производитель автомобиля | varchar |
| model | Модель автомобиля | varchar |
| color | Цвет автомобиля | varchar |
| year | Год выпуска автомобиля | int |
| status | Состояние автомобиля | varchar |
| number | Номер автомобиля | varchar |
| is\_free | Свободна ли машина | boolean |
| price | Цена автомобиля за день | numeric |
| location\_id | Номер местоположения автомобиля | int |

Таблица Location представляет список всех местоположений(таблица 3.3).

Таблица 3.3 – Столбцы таблицы Location

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| location\_id | Идентификатор местоположения | int |
| address | Адрес | varchar |
| Geom | Координаты | Geometry |

Таблица Rental представляет список всех бронирований (таблица 3.4).

Таблица 3.4 – Столбцы таблицы Rental

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| rental\_id | Идентификатор бронирования | int |
| start\_date | Дата начала бронирования | date |
| end\_date | Дата конца бронирования | date |
| total\_cost | Сумма заказа | numeric |
| car\_id | Идентификатор автомобиля | int |
| user\_id | Идентификатор пользователя | int |

Таблица Node представляет список всех узлов (таблица 3.5).

Таблица 3.5 – Столбцы таблицы Node

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| id | Идентификатор узла | int |
| data | Данные узла | jsonb |

Таблица Edge представляет список всех ребер (таблица 3.6).

Таблица 3.6 – Столбцы таблицы Edge

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| id | Идентификатор ребра | bigint |
| next\_node\_id | Идентификатор предыдущего узла | int |
| previous\_node\_id | Идентификатор следующего узла | int |

* 1. Пользователи и роли

В базе данных приложения есть одна основная таблица пользователей: User. В этой таблице есть поле `role`, которое определяет роль пользователя в системе. Всего в системе предусмотрено две роли: 'admin' и 'user'.

Администратор имеет полный доступ ко всем операциям в приложении. Он может управлять всеми данными в базе данных, включая автомобили, бронирования, местоположения и пользователей.

Пользователь имеет ограниченный доступ к операциям в приложении. Он может просматривать доступные автомобили, бронировать автомобили, просматривать свои бронирования и отменять свои бронирования.

Важно правильно управлять доступом к данным на основе роли пользователя, чтобы обеспечить безопасность и целостность вашей базы данных. Это также помогает обеспечить хороший пользовательский опыт, поскольку пользователи получают доступ только к тем функциям, которые им нужны.

* 1. Процедуры и функции

Вместо традиционного подхода с использованием пакетов и процедур в базе данных, приложение использует Django, фреймворк Python, для управления данными. Django предоставляет высокоуровневый API для взаимодействия с базой данных, который позволяет выполнять все необходимые операции без написания SQL-запросов.

Например, для получения всех пользователей используется User.objects.all(). Этот метод Django возвращает коллекцию объектов пользователей. Все эти операции выполняются внутри Django.

Таким образом, управление данными в приложении полностью осуществляется с помощью Django, что обеспечивает гибкость и удобство при работе с данными.

1. Описание процедур импорта и экспорта
   1. Процедура импорта данных в БД

Чтобы импортировать данный в БД был выбран формат XML.

Вместо традиционного подхода с использованием SQL и JSON для импорта данных, приложение использует Python и XML. Была разработана функция import\_streets\_from\_xml, которая считывает данные о улицах из XML-файла и сохраняет их в базе данных.

Код функции import\_streets\_from\_xml представлен в листинге 4.1.

def import\_streets\_from\_xml():

current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

file\_path = os.path.join(current\_dir, 'static', 'main', 'xml', 'streetsWithCoordinates.xml')

tree = ET.parse(file\_path)

root = tree.getroot()

for street in root.findall('street'):

lat = street.get('lat')

lon = street.get('lon')

address = street.text

location = Location(address=address)

location.geom = f'POINT({lon} {lat})'

location.save()

Листинг 4.1 – Функция import\_streets\_from\_xml

* 1. Процедура экспорта данных из БД

Для экспорта данных из базы данных также используется Python и Django. Была разработана функция export\_data\_to\_xml, которая извлекает все объекты из таблиц базы данных, преобразует их в XML-представление и сохраняет их в отдельные файлы.

Код функции export\_data\_to\_xml представлен в листинге 4.2.

def export\_data\_to\_xml():

# Получаем все объекты из таблиц

locations = Location.objects.all()

cars = Car.objects.all()

users = User.objects.all()

rentals = Rental.objects.all()

# Преобразуем объекты в XML-представление и сохраняем их в отдельные файлы

current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

locations\_file\_path = os.path.join(current\_dir, 'static', 'main', 'xml', 'locations.xml')

cars\_file\_path = os.path.join(current\_dir, 'static', 'main', 'xml', 'cars.xml')

users\_file\_path = os.path.join(current\_dir, 'static', 'main', 'xml', 'users.xml')

rentals\_file\_path = os.path.join(current\_dir, 'static', 'main', 'xml', 'rentals.xml')

with codecs.open(locations\_file\_path, 'w', encoding='utf-8') as locations\_file:

locations\_xml = serialize('xml', locations)

locations\_file.write(locations\_xml)

with codecs.open(cars\_file\_path, 'w', encoding='utf-8') as cars\_file:

cars\_xml = serialize('xml', cars)

cars\_file.write(cars\_xml)

with codecs.open(users\_file\_path, 'w', encoding='utf-8') as users\_file:

users\_xml = serialize('xml', users)

users\_file.write(users\_xml)

with codecs.open(rentals\_file\_path, 'w', encoding='utf-8') as rentals\_file:

rentals\_xml = serialize('xml', rentals)

rentals\_file.write(rentals\_xml)

print('Data exported successfully')

Листинг 4.2 – Функция export\_data\_to\_xml

1. Тестирование производительности

Для тестирования производительности БД в таблицу Rental было добавлено 100000 строк при помощи функции генерации. Для того что бы провести тесты цены запроса и времени его выполнения были разработаны три select запроса затрагивающие выборку с ключевыми полями и без них. Тестирование производиться при помощи инструментов pgAdmin.

Первый запрос, подремонтированный в листинге 5.1, необходим для теста стоимости и времени запроса при поиске строк с указанной суммой заказа.

SELECT \* FROM rental where total\_cost > 500

Листинг 5.1 – Первый тестовый запрос

Результат первого тестового запроса показал, что время запроса составляет около 16ms, а среднее значение цены 993 что является не плохим результатом для поиска без ключевых или же индексированных полей в БД с 100000 строк. Результат выполнения предоставлен на рисунке 5.1.

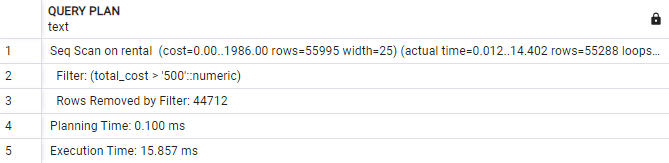


Рисунок 5.1 – Результат выполнения первого тестового запроса

Второй запрос, подремонтированный в листинге 5.2, необходим для теста стоимости и времени запроса при поиске строки с указанной почтой.

SELECT \* FROM main\_rental WHERE end\_date > '2023-12-25'

Листинг 5.2 – Второй тестовый запрос

Результат второго тестового запроса показал, что время запроса составляет около 12ms, что является хорошим результатом для поиска с ключевым или же индексированным полем в БД с 100000 строк. Результат выполнения предоставлен на рисунке 5.2.

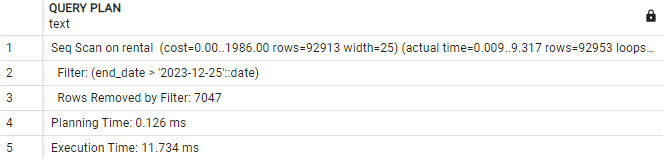


Рисунок 5.2 – Результат выполнения второго тестового запроса

Третий запрос, подремонтированный в листинге 5.3, необходим для теста стоимости и времени запроса при поиске строк с указанным идентификатором заказа. Данный тест является важным ведь в приложении часто используется подобный поиск.

SELECT \* FROM rental WHERE rental\_id = 56112

Листинг 5.3 – Третий тестовый запрос

Результат третьего тестового запроса показал, что время запроса составляет около 0.036ms, что является не плохим результатом для поиска без ключевых или же индексированных полей в БД с 100000 строк. Результат выполнения предоставлен на рисунке 5.3.

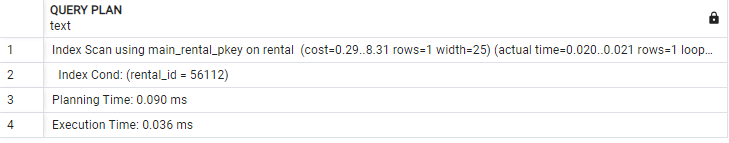


Рисунок 5.3 – Результат выполнения третьего тестового запроса

Однако было принято решение оптимизировать именно этот тип запросов так как он является самым частым по отношению к данной таблице. Был разработан простой индекс код, которого предоставлен в листинге 5.4.

CREATE INDEX RENTALID\_INDEX ON rental (rental\_id);

Листинг 5.4 – Индекс для таблицы Rental

Повторной результат третьего тестового запроса показал, что время запроса практически не снизилось и составляет 0.032ms, что является хорошим результатом для поиска с ключевым или же индексированным полем в БД с 100000 строк. Результат выполнения предоставлен на рисунке 5.4.

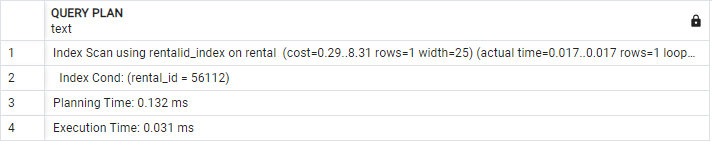


Рисунок 5.4 – Повторный результат выполнения третьего тестового запроса

По результатам проведённых в рамках тестирования производительности базы данных были проведены некоторые работы, направленные оптимизировать производительность. После проведения повторных тестов можно заявлять, что база данных является достаточно производительной что бы не вызывать проблем у пользователя, а также что база данных выдерживает 100000 строк без сильных падений в производительности.

1. Описание технологии применяемой в БД

В качестве основных технологий в БД были выбраны шифрование и маскирование. Шифрование включает в себя шифровку некоторых данных в целях скрыть данные с возможность восстановления или сопоставления. Маскирование включает в себя частичное сокрытие некоторых данных в целях показать часть информации, но не показывать её полностью.

* 1. Шифрование

Для реализации шифрование в курсовом проекте были использован встроенный инструмент Oracle «dbms\_crypto» предоставляющий возможности для шифрования и хеширования. Для шифрования был использован метод «dbms\_crypto.ENCRYPT» который шифрует входящую строку. Для шифрования ему необходимо указать входную строку в raw формате, алгоритм шифрования и ключ в виде строки в raw формате. Для шифрования был выбран ключ «MEDKIT» и алгоритм шифрования RC4. Алгоритм шифрования RC4 (Rivest Cipher 4) является симметричным потоковым шифром, который был разработан Роном Ривестом в 1987 году. RC4 используется для шифрования и дешифрования данных, и он особенно известен своей простотой и эффективностью. Процесс шифрования предоставлен в листинге 6.1.

PROCEDURE REGISTRATION\_NEW\_USER(

…,

password\_a in varchar2,

…

)

AS

v\_encrypted\_raw raw(128);

BEGIN

v\_encrypted\_raw := dbms\_crypto.ENCRYPT(

utl\_i18n.string\_to\_raw(password\_a, 'AL32UTF8'),

DBMS\_CRYPTO.ENCRYPT\_RC4,

utl\_i18n.string\_to\_raw('MEDKIT')

);

-- INSERT INTO USERS (…) VALUES (…, v\_encrypted\_raw, …);

commit;

end REGISTRATION\_NEW\_USER;

Листинг 6.1 – Процесс шифрования пароля.

Шифрование, как видно по листингу 6.1, было выбрано для сокрытия пароля пользователя. Так же шифрование используется для авторизации. Когда в процедуру для авторизации приходит пароль он шифруются и в последствии сопоставляются шифры если они одинаковые соответственно тогда пароли тоже одинаковые из-за чего авторизация разрешается.

* 1. Маскирование

Для реализации маскирования были использованы встроенные функции для работы со строками Oracle. Для маскирования используется встроенная функция «SUBSTR» принимающая в качестве параметров входную строку, начальный индекс символа и конечный индекс символа. Потом производиться конкатенация с строкой «\*\*\*\*\*\*\*» в итоге чего выходит замаскированный телефонный номер. Процесс маскирования предоставлен в листинге 6.2.

CREATE OR REPLACE FUNCTION GET\_ALL\_USERS

RETURN SYS\_REFCURSOR

IS

v\_result SYS\_REFCURSOR;

BEGIN

OPEN v\_result FOR

SELECT USER\_ID, USER\_ROLE, NAME, SURNAME, PATRONYMIC,

'HIDDEN' AS PASSWORD,

BIRTHDAY,

SUBSTR(PHONE\_NUMBER, 0, 7) || '\*\*\*\*\*\*\*' AS PHONE\_NUMBER,

EMAIL FROM USERS;

RETURN v\_result;

end;

Листинг 6.2 – Процесс маскирования телефонного номера пользователя

Маскирование используется для частичного сокрытия телефонного номера. Выбирается первые 8 символов, а остальное заполняется звёздами.

1. Краткое описание приложения

Для демонстрации работы БД было разработано приложение, написанное на языке java при помощи фреймворка Spring boot с использование JavaScript. Это веб-приложение с использованием многоуровневой архитектуры MVC. Так же в приложении используются разные паттерны проектирования такие как: repository, singleton, factory, unit of work и т.д. Для с БД были разработаны специальные классы.

Класс SessionManager и SessionInstance созданы для синхронизации сессии БД и веб. SessionManager создаёт сессию БД под необходимым пользователем, этими пользователями могут быть все ранее описанные пользователи: ADMIN, DOCTOR, PATIENT, LOGREG. ADMIN в приложении это пользователь «SYS AS SYSDBA» в БД. Так же SessionManager хранит все созданные сессии в виде словаря, где ключ – это идентификатор сессии веб, а значение это экземпляр класс SessionInstance.

SessionInstance - это класс хранящий в себе соединение конкретного пользователя с БД, а так же является реализацией паттерна unit of work и хранит в себе все репозитории необходимые для взаимодействия с БД. Для взаимодействия с пакетами в БД был реализован паттерн repository который хранит в себе все необходимые методы для вызова функций и процедур из соответствующих пакетов. Код интерфейса, который реализуют все классы для обращения к таблицам предоставлен в листинге 7.1.

public interface OracleRepository<T> {

void insert(T element) throws SQLException;

void delete(T element) throws SQLException;

void update(T element) throws SQLException;

T get(T element) throws SQLException;

T getById(int id) throws SQLException;

List<T> getAll() throws SQLException;

}

Листинг 7.1 – Интерфейс репозитория

Для сопоставления данных из БД в приложение были созданные специальные классы модели. Для того чтобы была возможность управлять связями многие ко многим был разработан специальный класс сервис OracleMTMConnectionService. Класс OracleMTMConnectionService предоставляет методы для обращения к процедурам и функциям из пакета MTM\_PACK.

1. Руководство пользователя

Разработанное в рамках курсового проекта приложение поддерживает три основные роли: администратор, врач и пациент. Заходя на главную страницу приложения пользователю представляется форма для авторизации, а также кнопки отправки формы и перехода к странице регистрации. Скриншот страницы авторизации предоставлен на рисунке 8.1.

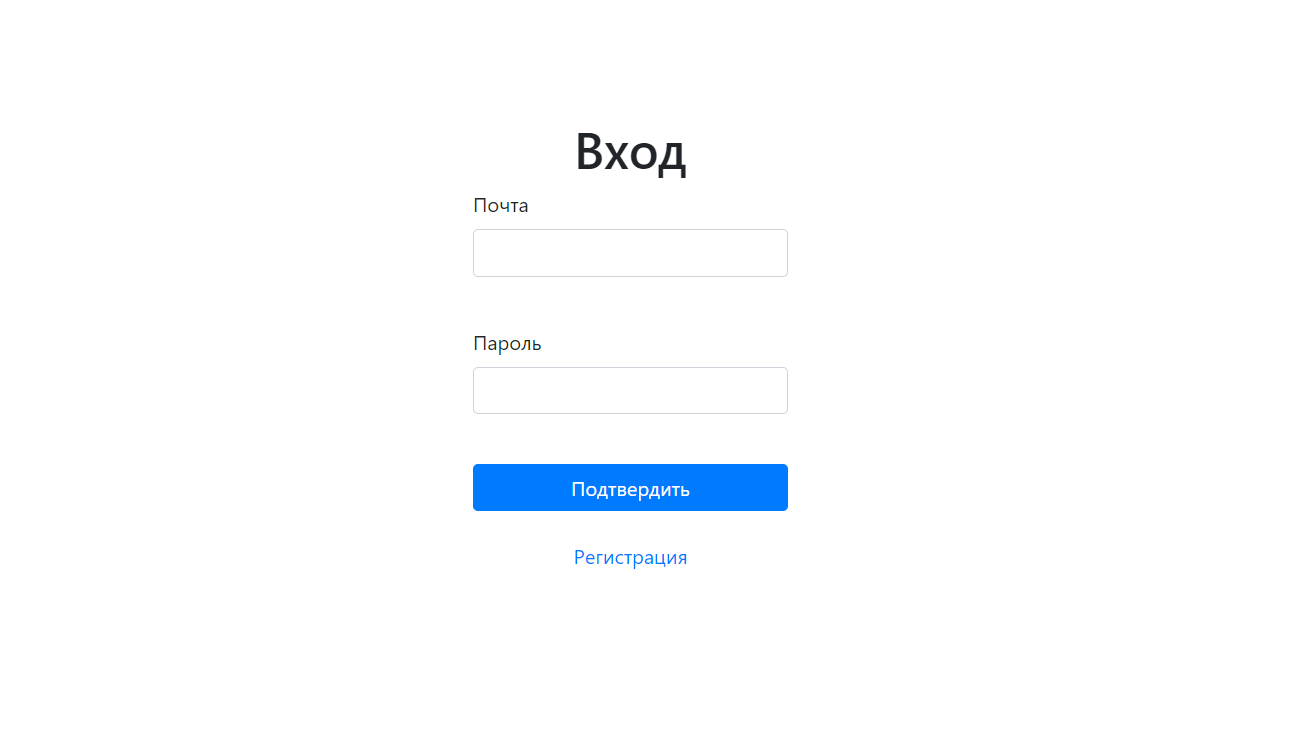


Рисунок 8.1 – Страница авторизации

После авторизации пользователь попадёт на основную страницу, где будет выведена информация о нем. В зависимости от роли у пользователя будет разный набор возможный действий.

Администратору доступны обозреватели болезней, симптомов, лекарств, пользователей. Скриншот основной страницы админа приведена на рисунке 8.2

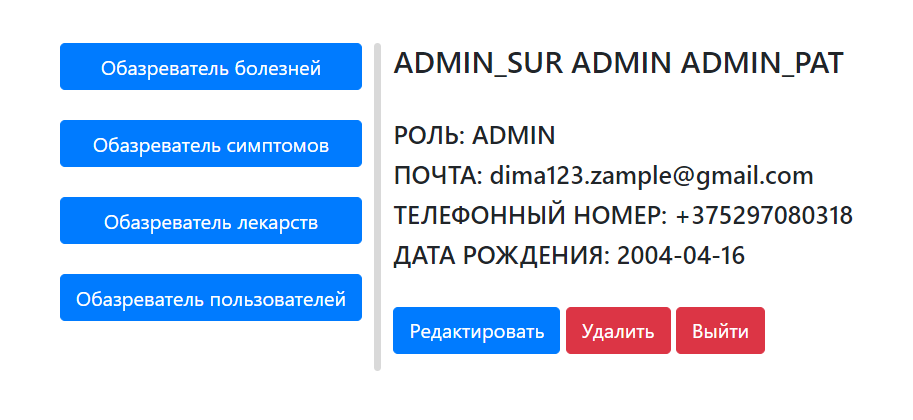


Рисунок 8.2 – Главная страница админа

В каждом обозревателе есть возможность добавлять, изменять, удалять элементы. В пример приведён обозреватель лекарств на рисунке 8.3.

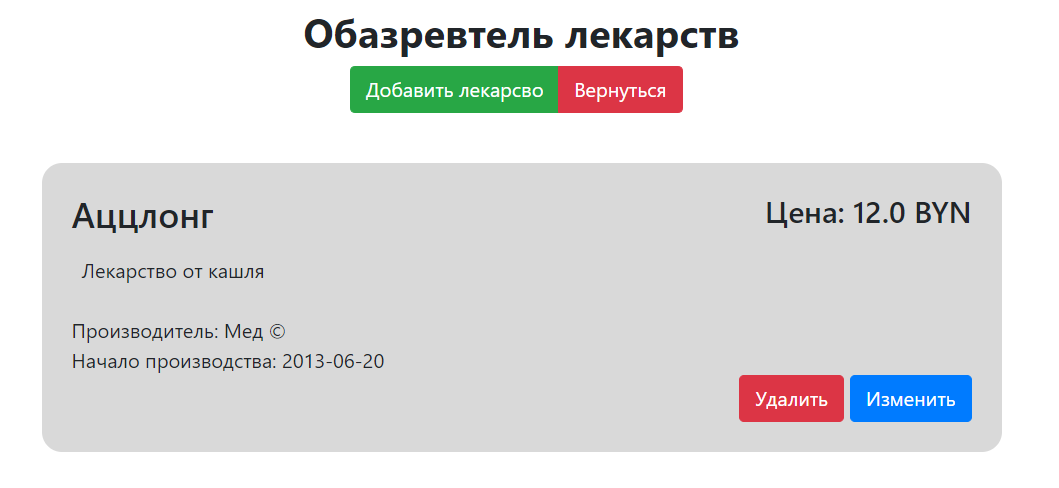


Рисунок 8.3 – Обозреватель лекарств

Для изменения или добавления элементов существую специальные редакторы в которых требуется заполнить специальные поля, некоторые поля проходят валидацию и их можно заполнить только специальными данными. Редактор лекарств приведён на рисунке 8.4.

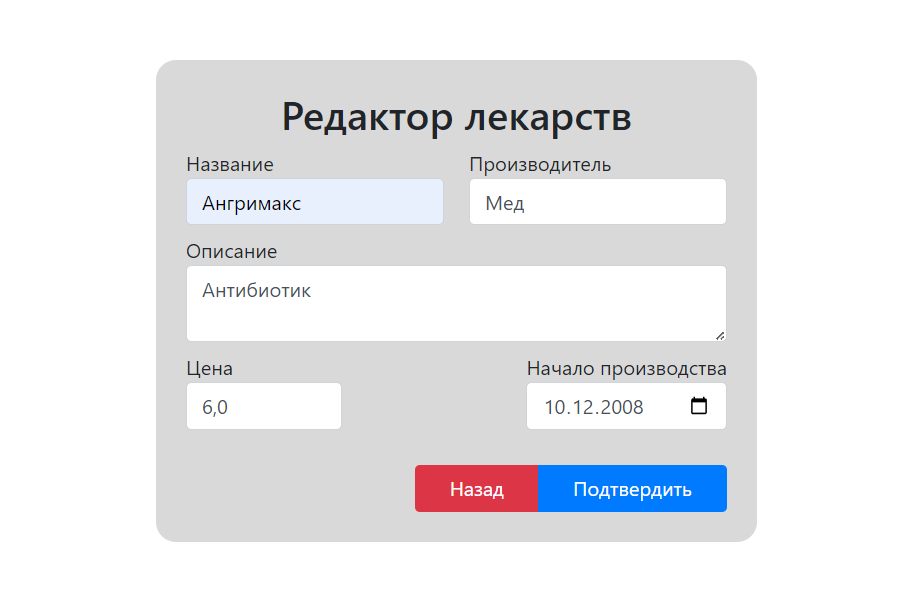


Рисунок 8.4 – Редактор лекарств

В этом редакторе требуется ввод следующих данных: название, производитель, описание, цена, начало производства. Все поля проходят валидацию на наличие данных внутри. В поле даты проверяется что бы не была указана бедующая дата.

Если пользователь авторизируется с ролью врача, то он сможет создавать диагноз, просматривать текущие диагнозы, просматривать текущие приёмы, просматривать все диагнозы, которые он выставлял, просматривать все посещение которые у него были. Скриншот главной страницы врача предоставлен на рисунке 8.5.

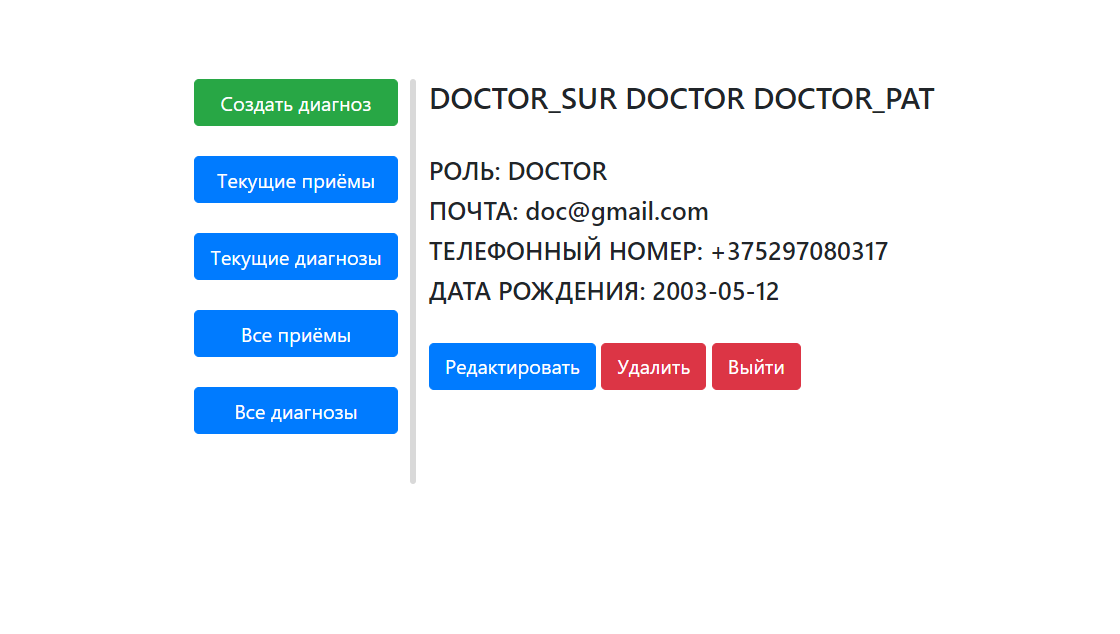


Рисунок 8.5 – Главная страница врача

Если пользователь авторизируется с ролью пациента, то он сможет записываться на приём, просматривать все свои приёмы, просматривать медицинскую карту. Медицинская карта – это список из диагнозов, которые есть у пользователя. Скриншот главной страницы пациента предоставлен на рисунке 8.6.



Рисунок 8.6 – Главная страница пациента

Так же всем пользователем в главной странице доступны кнопки редактирования и удаления своего аккаунта.

Заключение

Результатом выполнения курсового проекта стало вэб-приложение с использованием технологий Java Spring boot для взаимодействия с базой данных. В ходе курсового проекта была использована база данных Oracle, которая спроектирована для поликлиники.

Были реализованы все основные требования, указанные в листе задания вместе со следующими пунктами:

* использование архитектурных шаблонов проектирования;
* использование системы управления базами данных (СУБД);
* доступ к данным должен осуществляться только через соответствующие хранимые процедуры.

Возможности администратора:

* управление базой данных болезней;
* управление базой данных симптомов;
* управление базой данных лекарств;
* управление базой данных а.  
   Возможности врача:
* подтверждение/отмена посещения;
* создание/редактирование диагноза;
* изменение состояния диагноза;
* просмотр активных посещений;
* просмотр активных диагнозов.

Возможности пациента:

* запись на посещение;
* отмена посещения;
* просмотр своих диагнозов;
* просмотр своих посещений.

Функциональные требования программного средства:

* вся информация должна храниться в базе данных;
* приложение должно производить валидацию вводимых пользователем различных 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.

Данный курсовой проект является примером разработки базы данных и приложения для работы с ней. Протестировав все компоненты приложения, можно прийти к заключению, что все основные требования выполнены и приложение работает исправно.

Список используемых источников

1. METANIT.COM Сайт о программировании [Электронный ресурс] Режим доступа: https://metanit.com – Дата доступа: 10.12.2023.
2. Oracle-dba.ru [Электронный ресурс] / Режим доступа: <https://oracle-dba.ru> – Дата доступа: 10.12.2023.
3. Шифрование в PL/SQL: уроки программирования [Электронный ресурс] / Режим доступа: https://oracle-patches.com/db/sql/3939-shifrovanie-v-pl-sql-uroki-programmirovaniya – Дата доступа: 10.12.2023.
4. Developing and Using Stored Procedures [Электронный ресурс] / Режим доступа: https://docs.oracle.com/cd/B2835901/appdev.111/b28843/tdddgprocedures.htm – Дата доступа: 10.12.2023.
5. Stackoverflow.com [Электронный ресурс] / Режим доступа: https://stackoverflow.com – Дата доступа: 10.12.2023.

Приложение А

create table Users (  
 user\_id int generated always

as identity (start with 1 increment by 1),  
 primary key (user\_id),  
  
 user\_role nvarchar2(32) not null,  
 name nvarchar2(64) not null,  
 surname nvarchar2(64) not null,  
 patronymic nvarchar2(64) not null,  
 password nvarchar2(128) not null,  
 birthday date not null,

email nvarchar2(64) not null unique  
) tablespace MEDKIT\_TS;  
  
create table Appointments (  
 appointment\_id int generated always as identity (start with 1 increment by 1),  
 primary key (appointment\_id),  
  
 doctor\_id int not null,  
 foreign key (doctor\_id) references Users(user\_id),  
  
 patient\_id int not null,  
 foreign key (patient\_id) references Users(user\_id),  
  
 appointment\_date date not null,  
 appointment\_state int default(0)  
) tablespace MEDKIT\_TS;

create table Diseases (  
 disease\_id int,  
 primary key (disease\_id),  
  
 name nvarchar2(64) not null,  
 description nvarchar2(512)  
) tablespace MEDKIT\_TS;

create table Diagnoses (  
 diagnose\_id int,  
 primary key (diagnose\_id),  
  
 doctor\_id int not null,  
 foreign key (doctor\_id) references Users(user\_id),  
  
 patient\_id int not null,  
 foreign key (patient\_id) references Users(user\_id),  
  
 disease\_id int,  
 foreign key (disease\_id) references Diseases(disease\_id),  
  
 open\_date date not null,  
 close\_date date,  
 note nvarchar2(128),  
 description nvarchar2(512),  
 diagnose\_state int default(0)  
) tablespace MEDKIT\_TS;

create table Symptoms (  
 symptom\_id int generated always as identity (start with 1 increment by 1),  
 primary key (symptom\_id),  
  
 name nvarchar2(64) not null,  
 description nvarchar2(256)  
) tablespace MEDKIT\_TS;  
  
create table Medicines (  
 medicine\_id int generated always as identity (start with 1 increment by 1),  
 primary key (medicine\_id),  
  
 name nvarchar2(64) not null,  
 description nvarchar2(512),  
 manufacturer nvarchar2(64) not null,  
 price number(12, 4) default(0),  
 start\_date date not null  
) tablespace MEDKIT\_TS;  
  
create table DiagnosesToMedicines (  
 id int generated always as identity (start with 1 increment by 1),  
 primary key (id),  
  
 diagnose\_id int not null,  
 foreign key (diagnose\_id) references Diagnoses(diagnose\_id),  
  
 medicine\_id int not null,  
 foreign key (medicine\_id) references Medicines(medicine\_id)  
) tablespace MEDKIT\_TS;

create table DiagnosesToSymptoms (  
 id int generated always as identity (start with 1 increment by 1),  
 primary key (id),  
  
 diagnose\_id int not null,  
 foreign key (diagnose\_id) references Diagnoses(diagnose\_id),  
  
 symptom\_id int not null,  
 foreign key (symptom\_id) references Symptoms(symptom\_id)  
) tablespace MEDKIT\_TS;  
  
create table DiseasesToSymptoms (  
 id int generated always as identity (start with 1 increment by 1),  
 primary key (id),  
  
 disease\_id int not null,  
 foreign key (disease\_id) references Diseases(disease\_id),  
  
 symptom\_id int not null,  
 foreign key (symptom\_id) references Symptoms(symptom\_id)  
) tablespace MEDKIT\_TS;

Приложение Б

Пакет USER\_PACK

|  |  |  |  |
| --- | --- | --- | --- |
| Название процедуры или функции | Название параметра | Тип параметра | IN / OUT |
| REGISTRATION\_NEW\_USER | user\_role\_a | nvarchar2 | IN |
| name\_a | nvarchar2 | IN |
| surname\_a | nvarchar2 | IN |
| patronymic\_a | nvarchar2 | IN |
| password\_a | nvarchar2 | IN |
| birthday\_a | date | IN |
| phone\_number\_a | nvarchar2 | IN |
| email\_a | nvarchar2 | IN |
| GET\_ALL\_USERS | - | SYS\_REFCURSOR | OUT |
| GET\_CURRENT\_USER | - | SYS\_REFCURSOR | OUT |
| a\_email | nvarchar2 | IN |
| a\_password | nvarchar2 | IN |
| UPDATE\_CURRENT\_USER | user\_id\_a | int | IN |
| name\_a | nvarchar2 | IN |
| surname\_a | nvarchar2 | IN |
| patronymic\_a | nvarchar2 | IN |
| birthday\_a | date | IN |
| phone\_number\_a | nvarchar2 | IN |
| DELETE\_CURRENT\_USER | user\_id\_a | int | IN |
| GET\_USERS\_BY\_ROLE | - | SYS\_REFCURSOR | OUT |
| role\_a | nvarchar2 | IN |
| GET\_USER | - | SYS\_REFCURSOR | OUT |
| user\_id\_a | int | IN |
| GET\_USER\_BY\_EMAIL | - | SYS\_REFCURSOR | OUT |
| a\_email | nvarchar2 | IN |
| GET\_USERS\_BY\_NAME\_AND\_ROLE | - | SYS\_REFCURSOR | OUT |
| name\_a | nvarchar2 | IN |
| surname\_a | nvarchar2 | IN |
| patronymic\_a | nvarchar2 | IN |
| role\_a | nvarchar2 | IN |

Продолжение пакета USERS\_PACK

|  |  |  |  |
| --- | --- | --- | --- |
| Название процедуры или функции | Название параметра | Тип параметра | IN / OUT |
| GET\_FIRST\_USERS | - | SYS\_REFCURSOR | OUT |
| a\_count\_a | int | IN |
| GET\_USERS\_BY\_NAME | - | SYS\_REFCURSOR | OUT |
| name\_a | nvarchar2 | IN |
| surname\_a | nvarchar2 | IN |
| patronymic\_a | nvarchar2 | IN |

Пакет APPOINMENT\_PACK

|  |  |  |  |
| --- | --- | --- | --- |
| Название процедуры или функции | Название параметра | Тип параметра | IN / OUT |
| INSERT\_NEW\_APPOINTMENT | patient\_id\_a | int | IN |
| doctor\_id\_a | int | IN |
| date\_a | date | IN |
| UPDATE\_APPOINTMENT | appointment\_id\_a | int | IN |
| new\_state | int | IN |
| DELETE\_APPOINTMENT | appointment\_id\_a | int | IN |
| GET\_ALL\_APPOINTMENTS | - | SYS\_REFCURSOR | OUT |
| GET\_APPOINTMENT | - | SYS\_REFCURSOR | OUT |
| id\_a | int | IN |
| GET\_APPOINTMENTS\_BY\_USERID | - | SYS\_REFCURSOR | OUT |
| user\_id\_a | int | IN |

Пакет MEDICINE\_PACK

|  |  |  |  |
| --- | --- | --- | --- |
| Название процедуры или функции | Название параметра | Тип параметра | IN / OUT |
| INSERT\_NEW\_MEDICINE | name\_a | nvarchar2 | IN |
| description\_a | nvarchar2 | IN |
| manufacturer\_a | nvarchar2 | IN |
| price\_a | number | IN |
| start\_date\_a | date | IN |
| UPDATE\_MEDICINE | medicine\_id\_a | int | IN |
| name\_a | nvarchar2 | IN |
| description\_a | nvarchar2 | IN |
| manufacturer\_a | nvarchar2 | IN |
| price\_a | number | IN |
| start\_date\_a | date | IN |

Продолжение пакета MEDICINE\_PACK

|  |  |  |  |
| --- | --- | --- | --- |
| Название процедуры или функции | Название параметра | Тип параметра | IN / OUT |
| DELETE\_MEDICINE | medicine\_id\_a | int | IN |
| GET\_ALL\_MEDICINES | - | SYS\_REFCURSOR | OUT |
| GET\_MEDICINE | - | SYS\_REFCURSOR | OUT |
| id\_a | int | IN |

Пакет SYMPTOM\_PACK

|  |  |  |  |
| --- | --- | --- | --- |
| Название процедуры или функции | Название параметра | Тип параметра | IN / OUT |
| INSERT\_NEW\_SYMPTOM | name\_a | nvarchar2 | IN |
| description\_a | nvarchar2 | IN |
| UPDATE\_SYMPTOM | symptom\_id\_a | int | IN |
| name\_a | nvarchar2 | IN |
| description\_a | nvarchar2 | IN |
| DELETE\_SYMPTOM | symptom\_id\_a | int | IN |
| GET\_ALL\_SYMPTOMS | - | SYS\_REFCURSOR | OUT |
| GET\_SYMPTOM | - | SYS\_REFCURSOR | OUT |
| id\_a | int | IN |
| GET\_SYMPTOMS\_BY\_DISEASE | - | SYS\_REFCURSOR | OUT |
| disease\_id\_a | int | IN |

Пакет DIAGNOSE\_PACK

|  |  |  |  |
| --- | --- | --- | --- |
| Название процедуры или функции | Название параметра | Тип параметра | IN / OUT |
| INSERT\_NEW\_DIAGNOSE | doctor\_id\_a | int | IN |
| patient\_id\_a | int | IN |
| disease\_id\_a | int | IN |
| open\_date\_a | date | IN |
| close\_date\_a | date | IN |
| note\_a | nvarchar2 | IN |
| description\_a | nvarchar2 | IN |
| diagnose\_state\_a | int | IN |

Продолжение пакета DIAGNOSE\_PACK

|  |  |  |  |
| --- | --- | --- | --- |
| Название процедуры или функции | Название параметра | Тип параметра | IN / OUT |
| INSERT\_NEW\_DIAGNOSE\_RET\_ID | - | INT | OUT |
| doctor\_id\_a | int | IN |
| patient\_id\_a | int | IN |
| disease\_id\_a | int | IN |
| open\_date\_a | date | IN |
| close\_date\_a | date | IN |
| note\_a | nvarchar2 | IN |
| description\_a | nvarchar2 | IN |
| diagnose\_state\_a | int | IN |
| UPDATE\_DIAGNOSE | diagnose\_id\_a | int | IN |
| doctor\_id\_a | int | IN |
| patient\_id\_a | int | IN |
| disease\_id\_a | int | IN |
| open\_date\_a | date | IN |
| close\_date\_a | date | IN |
| note\_a | nvarchar2 | IN |
| description\_a | nvarchar2 | IN |
| diagnose\_state\_a | int | IN |
| DELETE\_DIAGNOSE | diagnose\_id\_a | int | IN |
| GET\_ALL\_DIAGNOSES | - | SYS\_REFCURSOR | OUT |
| GET\_DIAGNOSE | - | SYS\_REFCURSOR | OUT |
| id\_a | int | IN |
| GET\_DIAGNOSE\_BY\_USER | - | SYS\_REFCURSOR | OUT |
| user\_a | int | IN |
| ANALYSE\_DIAGNOSE | - | SYS\_REFCURSOR | OUT |
| JSON | CLOB | IN |

Пакет DISEASE\_PACK

|  |  |  |  |
| --- | --- | --- | --- |
| Название процедуры или функции | Название параметра | Тип параметра | IN / OUT |
| INSERT\_NEW\_DISEASE | name\_a | nvarchar2 | IN |
| description\_a | nvarchar2 | IN |
| INSERT\_NEW\_DISEASE\_RET\_ID | - | INT | OUT |
| name\_a | nvarchar2 | IN |
| description\_a | nvarchar2 | IN |
| UPDATE\_DISEASE | disease\_id\_a | int | IN |
| name\_a | nvarchar2 | IN |
| description\_a | nvarchar2 | IN |
| DELETE\_DISEASE | disease\_id\_a | int | IN |
| GET\_ALL\_DISEASES | - | SYS\_REFCURSOR | OUT |
| GET\_DISEASE | - | SYS\_REFCURSOR | OUT |
| id\_a | int | IN |

Пакет MTM\_PACK

|  |  |  |  |
| --- | --- | --- | --- |
| Название процедуры или функции | Название параметра | Тип параметра | IN / OUT |
| DIAGNOSES\_TO\_MEDICINES | diagnose | int | IN |
| medicine | int | IN |
| DIAGNOSES\_TO\_SYMPTOMS | diagnose | int | IN |
| symptom | int | IN |
| DISEASES\_TO\_SYMPTOMS | disease | int | IN |
| symptom | int | IN |
| GET\_DISEASESTOSYMPTOMS\_BY\_DISEASE | - | SYS\_REFCURSOR | OUT |
| diagnose | int | IN |
| GET\_DIAGNOSESTOSYMPTOMS\_BY\_DIAGNOSE | - | SYS\_REFCURSOR | OUT |
| diagnose | int | IN |
| GET\_DIAGNOSESTOMEDICINES\_BY\_DIAGNOSE | - | SYS\_REFCURSOR | OUT |
| diagnose | int | IN |
| DELETE\_DIAGNOSESTOMEDICINES\_BY\_DIAGNOSE | diagnose | int | IN |
| DELETE\_DIAGNOSESTOSYMPTOMS\_BY\_DIAGNOSE | diagnose | int | IN |
| DELETE\_DISEASESTOSYMPTOMS\_BY\_DISEASE | disease | int | IN |

Приложение В

CREATE OR REPLACE TRIGGER DELETE\_USER\_LINKS\_TRIGGER

BEFORE DELETE

ON ADMIN.USERS

FOR EACH ROW

BEGIN

DELETE FROM ADMIN.APPOINTMENTS

WHERE ADMIN.APPOINTMENTS.PATIENT\_ID = :OLD.USER\_ID OR ADMIN.APPOINTMENTS.DOCTOR\_ID = :OLD.USER\_ID;

DELETE FROM ADMIN.DIAGNOSES

WHERE ADMIN.DIAGNOSES.PATIENT\_ID = :OLD.USER\_ID OR ADMIN.DIAGNOSES.DOCTOR\_ID = :OLD.USER\_ID;

end;

CREATE OR REPLACE TRIGGER DELETE\_DIAGNOSE\_LINKS\_TRIGGER

BEFORE DELETE

ON ADMIN.DIAGNOSES

FOR EACH ROW

BEGIN

DELETE FROM ADMIN.DIAGNOSESTOSYMPTOMS

WHERE ADMIN.DIAGNOSESTOSYMPTOMS.DIAGNOSE\_ID = :OLD.DIAGNOSE\_ID;

DELETE FROM ADMIN.DIAGNOSESTOMEDICINES

WHERE ADMIN.DIAGNOSESTOMEDICINES.DIAGNOSE\_ID = :OLD.DIAGNOSE\_ID;

end;

CREATE OR REPLACE TRIGGER DELETE\_SYMPTOM\_LINKS\_TRIGGER

BEFORE DELETE

ON ADMIN.SYMPTOMS

FOR EACH ROW

BEGIN

DELETE FROM ADMIN.DIAGNOSESTOSYMPTOMS

WHERE ADMIN.DIAGNOSESTOSYMPTOMS.SYMPTOM\_ID = :OLD.SYMPTOM\_ID;

DELETE FROM ADMIN.DISEASESTOSYMPTOMS

WHERE ADMIN.DISEASESTOSYMPTOMS.SYMPTOM\_ID = :OLD.SYMPTOM\_ID;

end;

CREATE OR REPLACE TRIGGER DELETE\_DISEASE\_LINKS\_TRIGGER

BEFORE DELETE

ON ADMIN.DISEASES

FOR EACH ROW

BEGIN

DELETE FROM ADMIN.DISEASESTOSYMPTOMS

WHERE ADMIN.DISEASESTOSYMPTOMS.DISEASE\_ID = :OLD.DISEASE\_ID;

DELETE FROM ADMIN.DIAGNOSES

WHERE ADMIN.DIAGNOSES.DISEASE\_ID = :OLD.DISEASE\_ID;

end;

CREATE OR REPLACE TRIGGER DELETE\_MEDICINE\_LINKS\_TRIGGER

BEFORE DELETE

ON ADMIN.MEDICINES

FOR EACH ROW

BEGIN

DELETE FROM ADMIN.DIAGNOSESTOMEDICINES

WHERE ADMIN.DIAGNOSESTOMEDICINES.MEDICINE\_ID = :OLD.MEDICINE\_ID;

end;

DROP TRIGGER DELETE\_USER\_LINKS\_TRIGGER;

DROP TRIGGER DELETE\_DIAGNOSE\_LINKS\_TRIGGER;

DROP TRIGGER DELETE\_DISEASE\_LINKS\_TRIGGER;

DROP TRIGGER DELETE\_SYMPTOM\_LINKS\_TRIGGER;

DROP TRIGGER DELETE\_MEDICINE\_LINKS\_TRIGGER;

Приложение Г

SELECT \* FROM JSON\_DOCUMENT;

CREATE DIRECTORY JSON\_DIR AS 'C:\Users\1234\Desktop\JSON';

INSERT INTO JSON\_DOCUMENT (data) VALUES (bfilename('JSON\_DIR', 'IMPORT.json'));

INSERT INTO ADMIN.USERS (USER\_ROLE, NAME, SURNAME, PATRONYMIC, PASSWORD, BIRTHDAY, PHONE\_NUMBER, EMAIL)

SELECT jt.USER\_ROLE, jt.NAME, jt.SURNAME, jt.PATRONYMIC, jt.PASSWORD, TO\_DATE(jt.BIRTHDAY, 'YYYY-MM-DD'),

jt.PHONE\_NUMBER, jt.EMAIL

FROM JSON\_DOCUMENT j,

JSON\_TABLE(

j.data,

'$[\*]'

COLUMNS (

USER\_ROLE VARCHAR2(32) PATH '$.USER\_ROLE',

NAME VARCHAR2(64) PATH '$.NAME',

SURNAME VARCHAR2(64) PATH '$.SURNAME',

PATRONYMIC VARCHAR2(64) PATH '$.PATRONYMIC',

PASSWORD VARCHAR2(128) PATH '$.PASSWORD',

BIRTHDAY VARCHAR2(10) PATH '$.BIRTHDAY',

PHONE\_NUMBER VARCHAR2(32) PATH '$.PHONE\_NUMBER',

EMAIL VARCHAR2(64) PATH '$.EMAIL'

)

) jt;

COMMIT;