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 база данных поликлиники с применением технологии шифрование и маскирование»

Выполнил студент Окулич Дмитрий Юрьевич

(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](#_Toc153353371)

[1 Постановка задачи 4](#_Toc153353372)

[1.1 Аналитический обзор аналогов и литературных решений 4](#_Toc153353373)

[1.2 Анализ и разработка функциональных требований 6](#_Toc153353374)

[2 Проектирование базы данных 8](#_Toc153353375)

[3 Разработка необходимых объектов 11](#_Toc153353376)

[3.1 Таблицы 11](#_Toc153353377)

[3.2 Пользователи и роли 13](#_Toc153353378)

[3.3 Процедуры и функции 15](#_Toc153353379)

[3.3.1 Пакет USER\_PACK 15](#_Toc153353380)

[3.3.2 Пакет APPOINTMENT\_PACK 16](#_Toc153353381)

[3.3.3 Пакет MEDICINE\_PACK 16](#_Toc153353382)

[3.3.4 Пакет SYMPTOM\_PACK 17](#_Toc153353383)

[3.3.5 Пакет DIAGNOSE\_PACK 17](#_Toc153353384)

[3.3.6 Пакет DISEASE\_PACK 18](#_Toc153353385)

[3.3.7 Пакет MTM\_PACK 18](#_Toc153353386)

[3.4 Триггеры 19](#_Toc153353387)

[3.5 Последовательности 20](#_Toc153353388)

[4 Описание процедур импорта и экспорта 21](#_Toc153353389)

[4.1 Процедура импорта данных в БД 21](#_Toc153353390)

[4.2 Процедура экспорта данных из БД 21](#_Toc153353391)

[5 Тестирование производительности 22](#_Toc153353392)

[6 Описание технологии применяемой в БД 25](#_Toc153353393)

[6.1 Шифрование 25](#_Toc153353394)

[6.2 Маскирование 26](#_Toc153353395)

[7 Краткое описание приложения 27](#_Toc153353396)

[8 Руководство пользователя 28](#_Toc153353397)

[Заключение 31](#_Toc153353398)

[Список используемых источников 32](#_Toc153353399)

[Приложение А 33](#_Toc153353400)

[Приложение Б 37](#_Toc153353401)

[Приложение В 42](#_Toc153353402)

[Приложение Г 44](#_Toc153353403)



# Введение

Цель данной работы заключается в создании реляционной базы данных для веб-приложения поликлиники, которое позволит пациентам записываться на приемы, а врачам анализировать и вести учёт диагнозов.

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 Spring boot на языке Java и HTML c использованием JavaScript.

Для гарантированной безопасности пользователей приложения в курсовой работе применяется метод шифрования паролей перед их сохранением в базу данных. Так же будет использоваться маскирование некоторых данных что бы пользователю было доступна только часть информации.

Основные требования к приложению:

* Реализация ролей администратора, врача и пациента;
* открытие, закрытие диагноза;
* регистрация и логирование;
* оформление приёма;
* взаимодействие с БД при помощи хранимых процедур и функций.

Содержание данной пояснительной записки отражает этапы выполнения курсового проекта.

1. Постановка задачи

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

Цели приложения состоит в демонстрации возможности базы данных и предоставления удобного инструмента для поликлиники.

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 аналогичных приложений или web-сервисов для контроля базы данных поликлиники в открытом доступе не много. Поэтому было рассмотрено всего пара аналогов.

Мобильное приложение «BestDoctor» – приложение, предназначенное для записи к врачу, а также помогаем в выборе специалиста и предлагает оптимальные решения для проблем, связанных с здоровьем.

Интерфейса приведён на рисунке 1.1.

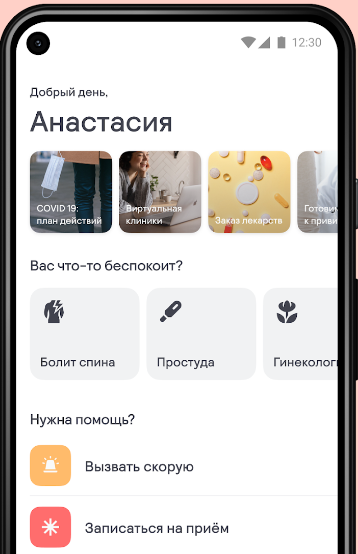


Рисунок 1.1 – Интерфейс приложения «BestDoctor»

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

Основные плюсы:

* удобный интерфейс;
* бесплатный;
* имеется чат для общения с врачом;
* имеется карта поликлиник.

Основные минусы:

* навязчивая реклама;
* мало функционала для врачей.

Мобильное приложение «справочник врача» – приложение является мобильным справочником для врачей с встроенным калькулятором анализов.

Интерфейс приведён на рисунке 1.2.

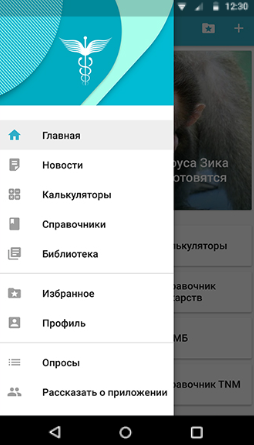


Рисунок 1.2 ­– интерфейс приложения «Справочник врача»

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

Основные плюсы:

* есть калькулятор анализов;
* большой объем справочников.

Основные минусы:

* неудобный интерфейс;
* нет функционала для пациентов.

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 хранимые процедуры.

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

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 болезней;
* управление базой данных симптомов;
* управление базой данных лекарств;
* управление базой данных пользователей.

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.

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.
* Должен производиться анализ диагноза по заданным симптомам.

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;
* посещение;
* болезнь;
* симптом;
* лекарство;
* диагноз.

Так же были определены необходимые связи. Например, между сущностями «пользователь» и «посещение» установлена связь один-ко-многим. Так же были выделены атрибуты для сущностей. Например, для сущности «пользователь» были выделены атрибуты такие, как «имя», «фамилия», «отчество», «дата рождения», «пароль», «мобильный номер», «роль».

Для взаимодействия с таблицами и приложением были определены специальные пользователи «администратор», «врач», «пациент» и служебный пользователь для авторизации и регистрации.

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.

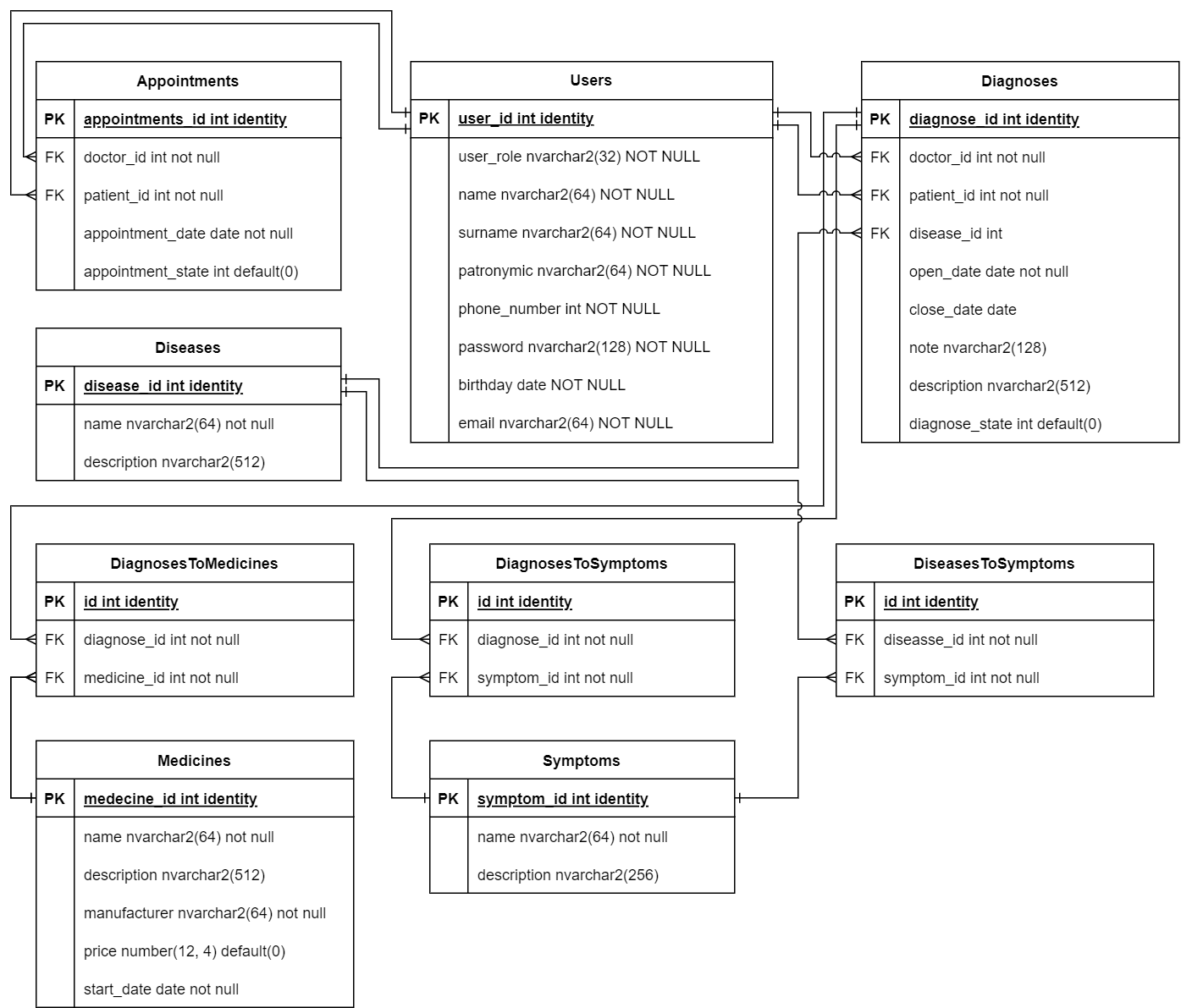


Рисунок 2.1 ­– Логическая модель базы данных

Всего в базе данных содержится 9 таблиц.

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

Таблица Appointments предназначена для хранения данных посещений.

Таблица Medicines предназначена для хранения данных лекарств.

Таблица Symptoms предназначена для хранения данных симптомов.

Таблица Diseases предназначена для хранения данных болезней.

Таблица Diagnoses предназначена для хранения данных диагнозов.

Таблица DiagnosesToMedicines предназначена для создания связи многие-ко-многим между таблицами Diagnoses и Medicines.

Таблица DiagnosesToSymptoms предназначена для создания связи многие-ко-многим между таблицами Diagnoses и Symptoms.

Таблица DiseasesToSymptoms предназначена для создания связи многие-ко-многим между таблицами Diseases и Symptoms.

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

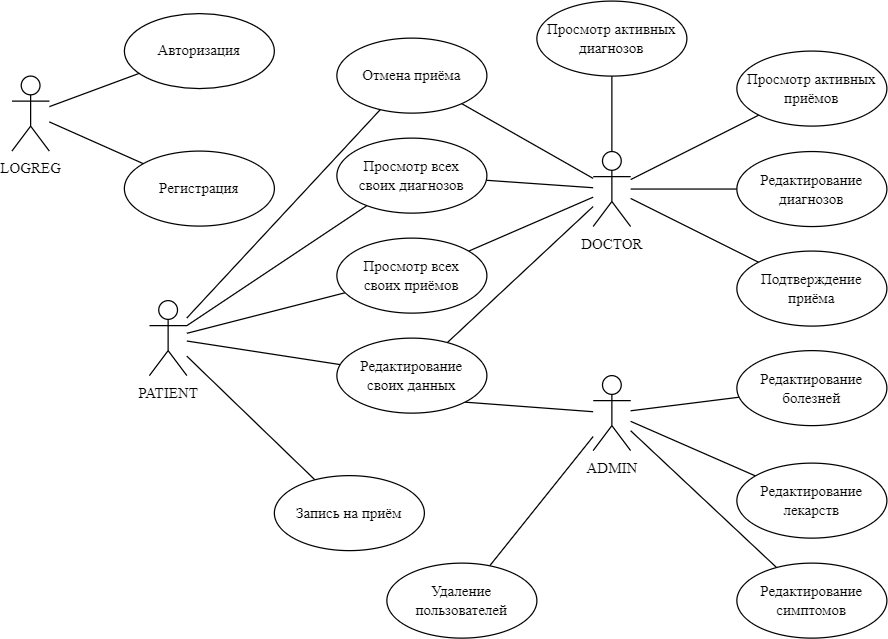


Рисунок 2.2 – Use-case диаграмма

Всего в базе данных 4 пользователя:

Пользователь LOGREG является служебным пользователем необходимым для авторизации и регистрации.

Пользователь ADMIN создан для представления администратора в базе данных. Данный пользователь имеет возможность редактировать болезни, лекарства, симптомы, удалять пользователей, редактировать свои данные.

Пользователь DOCTOR создан для представления врача в базе данных. Данный пользователь имеет возможность редактировать диагнозы, просматривать активные диагнозы, просматривать активные приёмы, просматривать все свои диагнозы, просматривать все свои приёмы, подтверждать приём, отменять приём, редактировать свои данные.

Пользователь PATIENT создан для представления администратора в базе данных. Данный пользователь имеет возможность просматривать свои диагнозы, просматривать свои приёмы, отменять приём, записываться на приём, редактировать свои данные.

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 база данных поликлиники с применением технологии шифрование и маскирование» было разработано 8 таблиц: Users, Appointments, Medicines, Symptoms, Diseases, DiagnosesToMedicines, DiagnosesToSymptoms, DiseasesToSymptoms, Diagnoses. Они будут реализованы в СУБД Oracle DataBase 19c.

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

Таблица 3.1 – Столбцы таблицы Users

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| user\_id | Идентификатор пользователя | int |
| user\_role | Роль пользователя | nvarchar2 |
| name | Имя пользователя | nvarchar2 |
| surname | Фамилия пользователя | nvarchar2 |
| patronymic | Отчество пользователя | nvarchar2 |
| phone\_number | Телефонный номер пользователя | int |
| password | Пароль пользователя | nvarchar2 |
| birthday | Дата рождения пользователя | date |
| email | Электронная почта пользователя | nvarchar2 |

Таблица Appointments представляет список всех посещений (таблица 3.2).

Таблица 3.2 – Столбцы таблицы Appointments

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| appointments\_id | Идентификатор посещения | int |
| doctor\_id | Идентификатор врача | int |
| patient\_id | Идентификатор пациента | int |
| appointments\_date | Дата посещения | date |
| appointments\_state | Состояние посещения | int |

Таблица Diseases представляет список всех болезней (таблица 3.3).

Таблица 3.3 – Столбцы таблицы Diseases

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| disease\_id | Идентификатор болезни | int |
| name | Название болезни | nvarchar2 |
| description | Описание болезни | nvarchar2 |

Таблица Diagnoses представляет список всех диагнозов (таблица 3.4).

Таблица 3.4 – Столбцы таблицы Users

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| diagnose\_id | Идентификатор диагноза | int |
| doctor\_id | Идентификатор врача | int |
| patient\_id | Идентификатор пациента | int |
| disease\_id | Идентификатор болезни | int |
| open\_date | Дата открытия диагноза | date |
| close\_date | Дата закрытия диагноза | date |
| note | Заметка к диагнозу | nvarchar2 |
| description | Описание диагноза | nvarchar2 |
| diagnose\_state | Состояние диагноза | int |

Таблица Medicines представляет список всех лекарств (таблица 3.5).

Таблица 3.5 – Столбцы таблицы Medicines

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| medicine\_id | Идентификатор лекарства | int |
| name | Название лекарства | nvarchar2 |
| description | Описание лекарства | nvarchar2 |
| manufacturer | Название производителя | nvarchar2 |
| price | Цена лекарства | number (12, 4) |
| start\_date | Дата начала производства | date |

Таблица Symptoms представляет список всех симптомов (таблица 3.6).

Таблица 3.6 – Столбцы таблицы Symptoms

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| symptom\_id | Идентификатор симптома | int |
| name | Название симптома | nvarchar2 |
| description | Описание симптома | nvarchar2 |

Таблица DiagnosesToMedicines представляет список связей между таблицами Diagnoses и Medicines (Многие-ко-многим) (Таблица 3.7).

Таблица 3.7 – Столбцы таблицы DiagnosesToMedicines

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| id | Идентификатор связи | int |
| diagnose\_id | Идентификатор диагноза | int |
| medicine\_id | Идентификатор лекарств | int |

Таблица DiagnosesToSymptoms представляет список связей между таблицами Diagnoses и Symptoms (Многие-ко-многим) (таблица 3.8).

Таблица 3.8 – Столбцы таблицы DiagnosesToSymptoms

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| id | Идентификатор связи | int |
| diagnose\_id | Идентификатор диагноза | int |
| symptoms\_id | Идентификатор симптома | int |

Таблица DiseasesToSymptoms представляет список связей между таблицами Diagnoses и Medicines (Многие-ко-многим) (таблица 3.8).

Таблица 3.8 – Столбцы таблицы DiseasesToSymptoms

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| id | Идентификатор связи | int |
| diseases\_id | Идентификатор болезни | int |
| medicine\_id | Идентификатор лекарств | int |

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

* 1. Пользователи и роли

База данных содержит 4 основных пользователя: ADMIN, DOCTOR, PATIENT и LOGREG. Пользователь ADMIN представляет собой хранилище всех объектов базы данных. Созданием инфраструктуры, объектов БД и управлением приложения занимается стандартный в Oracle пользователь SYS с привилегиями SYSDBA. Остальные пользователи могут взаимодействовать с базой данных благодаря полномочий на вызов процедур и функций из указанных пакетов.

Каждый пользователь имеет профиль, который представлен в листинге 3.1. Профиль позволяет наложить некоторые ограничения на использование пользователя. Так, например указано что может быть 50 предупреждений о неправильно входе, а количество пользователей не ограничено.

CREATE PROFILE COMMON\_PROFILE LIMIT

PASSWORD\_LIFE\_TIME 180

SESSIONS\_PER\_USER UNLIMITED

FAILED\_LOGIN\_ATTEMPTS 50

PASSWORD\_LOCK\_TIME 1

PASSWORD\_REUSE\_TIME 10

PASSWORD\_GRACE\_TIME DEFAULT

CONNECT\_TIME 600

IDLE\_TIME 120;

Листинг 3.1 – Профиль для всех пользователей

Для каждого пользователя была создана роль. Роль - поименованный набор привилегий. В пример создания роли предоставлен листинг 3.2 где создаётся роль для пользователя DOCTOR.

GRANT CREATE SESSION, RESTRICTED SESSION TO DOCTOR\_RL;

GRANT EXECUTE ON ADMIN.APPOINTMENT\_PACK TO DOCTOR\_RL;

GRANT EXECUTE ON ADMIN.DIAGNOSE\_PACK TO DOCTOR\_RL;

GRANT EXECUTE ON ADMIN.DISEASE\_PACK TO DOCTOR\_RL;

GRANT EXECUTE ON ADMIN.USER\_PACK TO DOCTOR\_RL;

GRANT EXECUTE ON ADMIN.MEDICINE\_PACK TO DOCTOR\_RL;

GRANT EXECUTE ON ADMIN.SYMPTOM\_PACK TO DOCTOR\_RL;

GRANT EXECUTE ON ADMIN.MTM\_PACK TO DOCTOR\_RL;

Листинг 3.2 – Создание роли для пользователя DOCTOR

После создания роли и назначения ей привилегий, необходимо создать пользователя и присвоить ему роль, созданную ранее. Создание пользователя DOCTOR приведено в листинге 3.3.

CREATE USER DOCTOR IDENTIFIED BY "DOC"

default tablespace MEDKIT\_TS

temporary tablespace MEDKIT\_TEMP\_TS

profile COMMON\_PROFILE

quota 32M on MEDKIT\_TS;

GRANT DOCTOR\_RL TO DOCTOR;

Листинг 3.3 – Создания пользователя DOCTOR

Остальные пользователи базы данных создаются по тому же принципу, но их ролям присваивается привилегия на вызов процедур и функций из других пакетов.

* 1. Процедуры и функции

Для управления данными, пользователи вызывают процедуры и функции, которые располагаются в пакетах. Пакет Oracle PL/SQL – это объект БД, который группирует логически связанные типы, элементы и подпрограммы.

В базе данных используется следующие пакеты: USER\_PACK, APPOINTEMNT\_PACK, MEDICINE\_PACK, SYMPTOM\_PACK, DISEASE\_PACK, MTM\_PACK. Описание пакетов представлено ниже, табличное описание функций приведено в приложении Б.

* + 1. Пакет USER\_PACK

Пакет USER\_PACK содержит 11 процедур и функций для работы с таблицей USERS, а также в процедурах и функциях этого пакета происходит шифрование.

Процедура REGISTRATION\_NEW\_USER необходима для регистрации нового пользователя. Процедура принимает 8 параметров: роль пользователя, имя, фамилию, отчество, пароль, дату дня рождения, телефонный номер и почту. Так же в этой процедуре происходит шифрование пароля при помощи алгоритма «RC4».

Функция GET\_ALL\_USERS необходима для возвращения всех пользователей в таблице в виде курсора.

Функция GET\_CURRENT\_USER необходима для авторизации пользователя. Функция принимает 2 параметра: почту и пароль. Функция шифрует пароль, находит пользователя с данной почтой и сравнивает шифры если они одинаковые значит пароль верный и тогда возвращается курсор с необходимым пользователем. Только это функция возвращает данные о пользователе в полном виде: с паролем и не замаскированным телефонным номером.

Процедура UPDATE\_CURRENT\_USER необходима для обновления информации об пользователе. Процедура принимает 6 параметров: идентификатор пользователя, имя, фамилию, отчество, дата дня рождения, телефонный номер. Процедура найдёт пользователя по идентификатору и заменит данные, указанные в параметрах процедуры.

Процедура DELETE\_CURRENT\_USER необходима для удаления пользователя из базы данных. Принимает в качестве параметра идентификатор пользователя. Удаляет пользователя с указанным идентификатор.

Функция GET\_USERS\_BY\_ROLE необходима для поиска пользователей с конкретной ролью. Принимает в качестве параметра роль. Возвращает курсор с пользователями из таблицы.

Функция GET\_USER необходима для поиска пользователя. Принимает в качестве параметра идентификатор пользователя. Возвращает курсор с пользователем из таблицы.

Функция GET\_USER\_BY\_MAIL необходим для поиска пользователя по заданной почте. Принимает в качестве параметра почту. Возвращает курсор с найденным пользователем.

Функция GET\_USERS\_BY\_NAME\_AND\_ROLE необходима для поиска пользователей по роли и имени. Принимает 4 параметра: имя, фамилию, отчество, роль. Возвращает курсор с пользователями, найденными по заданным параметрам.

Функция GET\_FIRST\_USERS необходима для выбора нескольких первых пользователей в таблице. Принимает в качестве параметра кол-во пользователей, которое будет выбрано. Возвращает курсор с выбранными пользователями.

Функция GET\_USER\_BY\_NAME необходим для поиска пользователей по заданной имени. Принимает 3 параметра: имя, фамилию, отчество. Возвращает курсор с найденным пользователями.

* + 1. Пакет APPOINTMENT\_PACK

Пакет APPOINTMENT\_PACK содержит 6 процедур и функций необходимых для работы с таблицей APPOINTMENTS.

Процедура INSERT\_NEW\_APPOINTMENT необходима для вставки нового элемента таблицы. Принимает 3 параметра: идентификатор пациента, идентификатор врача, дату посещения.

Процедура UPDATE\_APPOINTMENT необходима для обновления элемента таблицы. Принимает в качестве параметров идентификатор посещения и новое состояние в виде целого числа.

Процедура DELETE\_APPOINEMENT необходим для удаления элемента таблицы. Принимает в качестве параметра идентификатор посещения.

Функция GET\_ALL\_APPOINTMENTS необходим для получения всех элементов таблицы. Возвращает курсор со всеми элементами.

Функция GET\_APPOINTMENT необходим для получения конкретного элемента. В качестве параметра принимает идентификатор посещения. Возвращает курсор с найденным элементом.

Функция GET\_APPOINTMENTS\_BY\_USER идентификатор необходим для получения посещений, относящихся к конкретному пользователю. В качестве параметра принимает идентификатор пользователя. Возвращает курсор с найденными элементами.

* + 1. Пакет MEDICINE\_PACK

Пакет MEDICINE\_PACK содержит 5 процедур и функций необходимых для работы с таблицей MEDICINES

Процедура INSERT\_NEW\_MEDICINE необходима для вставки нового элемента таблицы. Принимает 5 параметра: название, описание, производителя, цену, дата начала выпуска.

Процедура UPDATE\_MEDICINE необходима для обновления элемента таблицы. Принимает 6 параметров: идентификатор лекарства, название, описание, производителя, цену, дата начала выпуска.

Процедура DELETE\_MEDICINE необходим для удаления элемента таблицы. Принимает в качестве параметра идентификатор лекарства.

Функция GET\_ALL\_MEDICINES необходим для получения всех элементов таблицы. Возвращает курсор со всеми элементами.

Функция GET\_MEDICINES необходим для получения конкретного элемента. В качестве параметра принимает идентификатор лекарства. Возвращает курсор с найденным элементом таблицы.

* + 1. Пакет SYMPTOM\_PACK

Пакет SYMPTOM\_PACK содержит 6 процедур и функций необходимых для работы с таблицей SYMPTOMS

Процедура INSERT\_NEW\_SYMPTOM необходима для вставки нового элемента таблицы. Принимает 2 параметра: название, описание.

Процедура UPDATE\_SYMPTOM необходима для обновления элемента таблицы. Принимает 3 параметров: идентификатор симптома, название, описание.

Процедура DELETE\_SYMPTOM необходим для удаления элемента таблицы. Принимает в качестве параметра идентификатор симптома.

Функция GET\_ALL\_SYMPTOM необходим для получения всех элементов таблицы. Возвращает курсор со всеми элементами.

Функция GET\_SYMPTOM необходим для получения конкретного элемента. В качестве параметра принимает идентификатор симптома. Возвращает курсор с найденным элементом таблицы.

Функция GET\_SYMPTOMS\_BY\_DISEASE необходим для получения конкретного симптомов принадлежащей указанной болезни. В качестве параметра принимает идентификатор болезни. Возвращает курсор с найденными симптомами.

* + 1. Пакет DIAGNOSE\_PACK

Пакет DIAGNOSE\_PACK содержит 7 процедур и функций необходимых для работы с таблицей DIANOSES

Процедура INSERT\_NEW\_DIAGNOSE необходима для вставки нового элемента таблицы. Принимает 8 параметров: идентификатор врача, идентификатор пациента, идентификатор болезни, дата открытия диагноза, дата закрытия диагноза, заметка, описание, состояние диагноза.

Функция INSERT\_NEW\_DIAGNOSE\_RET\_ID необходима для вставки нового элемента таблицы и возвращения его идентификатора. Принимает 8 параметров: идентификатор врача, идентификатор пациента, идентификатор болезни, дата открытия диагноза, дата закрытия диагноза, заметка, описание, состояние диагноза. Возвращает идентификатор нового созданного элемента в виде целого числа.

Процедура UPDATE\_DIAGNOSE необходима для обновления элемента таблицы. Принимает 9 параметров: идентификатор диагноза, идентификатор врача, идентификатор пациента, идентификатор болезни, дата открытия диагноза, дата закрытия диагноза, заметка, описание, текущее состояние диагноза.

Процедура DELETE\_DIAGNOSE необходим для удаления элемента таблицы. Принимает в качестве параметра идентификатор диагноза.

Функция GET\_ALL\_DIAGNOSES необходим для получения всех элементов таблицы. Возвращает курсор со всеми элементами.

Функция GET\_DIAGNOSE необходим для получения конкретного элемента. В качестве параметра принимает идентификатор диагноза. Возвращает курсор с найденным элементом таблицы.

Функция GET\_DIAGNOSE\_BY\_USER необходим для получения конкретного симптомов принадлежащий указанному пользователю. В качестве параметра принимает идентификатор пользователя. Возвращает курсор с найденными диагнозами из таблицы.

Функция ANALYSE\_DIAGNOSE необходим для анализа диагноза по симптомам. В качестве параметра принимает переменную типа CLOB. Возвращает курсор с болезнями которые соответствуют данным симптомам.

* + 1. Пакет DISEASE\_PACK

Пакет DISEASE\_PACK содержит 6 процедур и функций необходимых для работы с таблицей DISEASES.

Процедура INSERT\_NEW\_DISEASE необходима для вставки нового элемента таблицы. Принимает 2 параметра: название, описание.

Функция INSERT\_NEW\_DISEASE\_RET\_ID необходима для вставки нового элемента таблицы и возвращения его идентификатора. Принимает 2 параметров: название, описание. Возвращает идентификатор ново созданного элемента в виде целого числа.

Процедура UPDATE\_DISEASE необходима для обновления элемента таблицы. Принимает 3 параметров: идентификатор болезни, название, описание.

Процедура DELETE\_DISEASE необходим для удаления элемента таблицы. Принимает в качестве параметра идентификатор болезни.

Функция GET\_ALL\_DISEASE необходим для получения всех элементов таблицы. Возвращает курсор со всеми элементами.

Функция GET\_DISEASE необходим для получения конкретного элемента. В качестве параметра принимает идентификатор болезни. Возвращает курсор с найденным элементом таблицы.

* + 1. Пакет MTM\_PACK

Пакет MTM\_PACK содержит 9 процедур и функций необходимых для работы с служебными таблицами базы данных: DIAGNOSESTOMEDICINES, DIAGNOSESTOSYMPTOMS, DISEASESTOSYMPTOMS. Эти таблицы не обходимы для осуществления связи многие ко многим.

Процедура DIAGNOSES\_TO\_MEDICINES необходима для создания связи между диагнозом и лекарством. В качестве параметров принимает идентификатор диагноза и идентификатор лекарства.

Процедура DIAGNOSES\_TO\_SYMPTOMS необходима для создания связи между диагнозом и симптомом. В качестве параметров принимает идентификатор диагноза и идентификатор симптома.

Процедура DISEASES\_TO\_SYMPTOMS необходима для создания связи между болезнью и симптомом. В качестве параметров принимает идентификатор болезни и идентификатор симптома.

Функция GET\_DISEASESTOSYMPTOMS\_BY\_DISEASE необходима для получения связи между болезнью и симптомом по выбранной болезни. В качестве параметра принимает идентификатор болезни. Возвращает курсор со связями болезней и симптомов.

Функция GET\_DIAGNOSESTOSYMPTOMS\_BY\_DIAGNOSE необходима для получения связи между диагнозом и симптомом по выбранному диагнозу. В качестве параметра принимает идентификатор диагноза. Возвращает курсор со связями диагнозов и симптомов.

Функция GET\_DIAGNOSESTOMEDICINES\_BY\_DIAGNOSE необходима для получения связи между диагнозом и лекарством по выбранному диагнозу. В качестве параметра принимает идентификатор диагноза. Возвращает курсор со связями диагнозов и лекарств.

Процедура DELETE\_DIAGNOSESTOMEDICINES\_BY\_DIAGNOSE нужна для удаление всех связей между конкретным диагнозом и лекарствами. В качестве параметра принимает идентификатор диагноза.

Процедура DELETE\_DIAGNOSESTOSYMPTOMS\_BY\_DIAGNOSE нужна для удаление всех связей между конкретным диагнозом и симптомами. В качестве параметра принимает идентификатор диагноза.

Процедура DELETE\_DISEASESTOSYMPTOMS\_BY\_DISEASE нужна для удаление всех связей между конкретной болезнью и симптомами. В качестве параметра принимает идентификатор болезни.

* 1. Триггеры

Для решения задачи автоматического удаления связей в служебных таблицах, а также связей пользователей, были использованы триггеры. Триггер - именованный программный блок, выполняемый в ответ на происходящие в базе данных события.

Для реализации автоматического удаления связей были созданы 5 триггеров: DELETE\_USER\_LINKS\_TRIGGER, DELETE\_DIAGNOSE\_LINKS\_TRIGGER, DELETE\_DISEASE\_LINKS\_TRIGGER, DELETE\_SYMPTOM\_LINKS\_TRIGGER, DELETE\_MEDICINE\_LINKS\_TRIGGER. Реализация этих триггеров приведена в приложении В.

Триггер DELETE\_USER\_LINKS\_TRIGGER необходим для автоматического удаления связей пользователя. Срабатывает до удаления элемента из таблицы USERS для каждой строки. Он занимается подчисткой таблиц APPOINTMENS и DIAGNOSES удаляя элементы, связанные с конкретным пользователем.

Триггер DELETE\_DIAGNOSE\_LINKS\_TRIGGER необходим для автоматического удаления связей диагноза. Срабатывает до удаления элемента из таблицы DIAGNOSES для каждой строки. Удаляет связи из служенных таблиц.

Триггер DELETE\_DISEASE\_LINKS\_TRIGGER необходим для автоматического удаления связей болезни. Срабатывает до удаления элемента из таблицы DISEASES для каждой строки. Удаляет связи из служенных таблиц, а также элементы таблицы DIAGNOSES.

Триггер DELETE\_SYMPTOM\_LINKS\_TRIGGER необходим для автоматического удаления связей симптома. Срабатывает до удаления элемента из таблицы SYMPTOMS для каждой строки. Удаляет связи из служенных таблиц.

Триггер DELETE\_MEDICINE\_LINKS\_TRIGGER необходим для автоматического удаления связей лекарства. Срабатывает до удаления элемента из таблицы MEDICINES для каждой строки. Удаляет связи из служенных таблиц.

* 1. Последовательности

Для генерации идентификаторов некоторых таблиц были использованы последовательности. Последовательность — это объект базы данных, который предоставляет уникальные числа (или последовательность), обычно используемые для генерации уникальных идентификаторов для столбцов с автоинкрементном в определённых таблицах.

Для генерации идентификаторов были созданы две последовательности: DISEASE\_ID\_SEQUENCE, DIAGNOSE\_ID\_SEQUENCE. Код создания последовательностей предоставлен в листинге 3.4.

CREATE SEQUENCE DISEASE\_ID\_SEQUENCE

MAXVALUE 10000000

START WITH 1

ORDER

CYCLE

NOCACHE;

CREATE SEQUENCE DIAGNOSE\_ID\_SEQUENCE

MAXVALUE 10000000

START WITH 1

CYCLE

NOCACHE;

Листинг 3.4 – Создание последовательностей

Генерация идентификаторов через последовательности в таблицах DISEASES и DIAGNOSES необходима по причине возвращения идентификатора нового элемента в функции вставки этих таблиц.

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

Чтобы импортировать данный в БД был выбран формат JSON.

Для импорта данных создается таблица с названием JSON\_DOCUMENT, которая содержит столбец data с типом данных clob и проверкой, что тип файла json. Код создания таблицы json\_documents представлен в листинге 4.1.

CREATE TABLE JSON\_DOCUMENT (

id int generated always as identity (start with 1 increment by 1),

data CLOB,

constraint json\_documents\_pk primary key (id),

constraint json\_documents\_json\_chk check (data is json)

) tablespace MEDKIT\_TS;

Листинг 4.1 – Создание таблицы JSON\_DOCUMENT

Затем необходимо создать директорию, в которой располагаются документы для импорта и загрузить их содержимое в таблицу JSON\_DOCUMENT. Затем извлечь необходимые данные в нужные таблицы. Этот процесс представлен в приложении Г.

* 1. Процедура экспорта данных из БД

Для экспорта данных из БД был выбран формат JSON. Для того что бы экспортировать данные из БД в JSON были использованы стандартные инструменты среды разработки IntelliJ DataGrip.

1. Тестирование производительности

Для тестирования производительности БД в таблицу USERS было добавлено 100000 строк при помощи импорта, описанного ранее. 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 среды разработки IntelliJ DataGrip.

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

SELECT \* FROM USERS WHERE USER\_ROLE = 'DOCTOR' AND SURNAME LIKE 'Оку%';

Листинг 5.1 – Первый тестовый запрос

Результат первого тестового запроса показал, что время запроса составляет около 50ms, а цена 651 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.

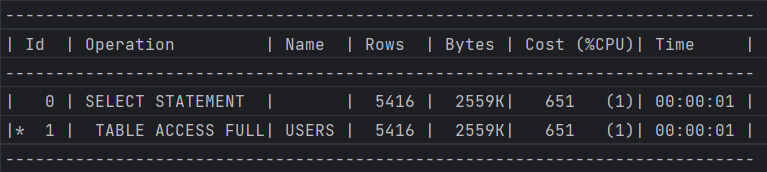


Рисунок 5.1 – Результат выполнения первого тестового запроса

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

SELECT \* FROM USERS WHERE EMAIL LIKE 'dima123.zample@gmail.com';

Листинг 5.2 – Второй тестовый запрос

Результат второго тестового запроса показал, что время запроса составляет около 27ms, а цена 2 что является хорошим результатом для поиска с ключевым или же индексированным полем в БД с 100000 строк. Результат выполнения предоставлен на рисунке 5.2.

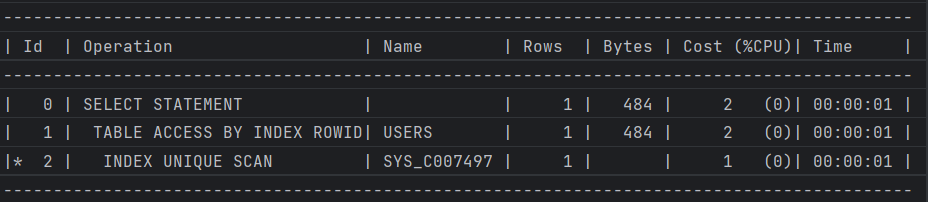


Рисунок 5.2 – Результат выполнения второго тестового запроса

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

SELECT \* FROM ADMIN.USERS

WHERE SURNAME LIKE 'Окулич' AND NAME LIKE 'Дмитрий'

AND PATRONYMIC LIKE 'Юрьевич';

Листинг 5.3 – Третий тестовый запрос

Результат третьего тестового запроса показал, что время запроса составляет около 61ms, а цена 652 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.

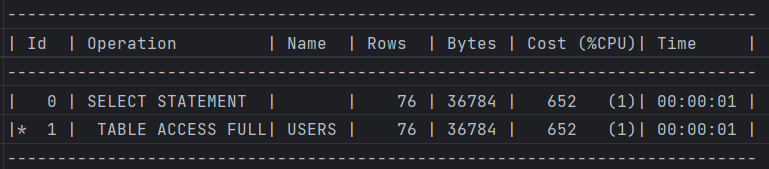


Рисунок 5.3 – Результат выполнения третьего тестового запроса

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 FULLNAME\_INDEX ON USERS (NAME, SURNAME, PATRONYMIC);

Листинг 5.4 – Индекс для таблицы USERS

Повторной результат третьего тестового запроса показал, что время запроса снизилось до 31ms, а цена составила 68 что является хорошим результатом для поиска с ключевым или же индексированным полем в БД с 100000 строк. Результат выполнения предоставлен на рисунке 5.4.

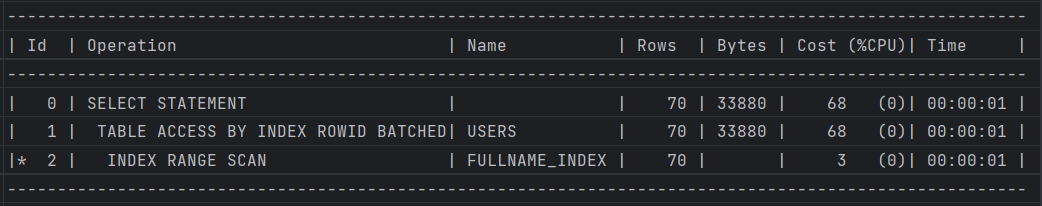


Рисунок 5.4 – Повторный результат выполнения третьего тестового запроса

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.

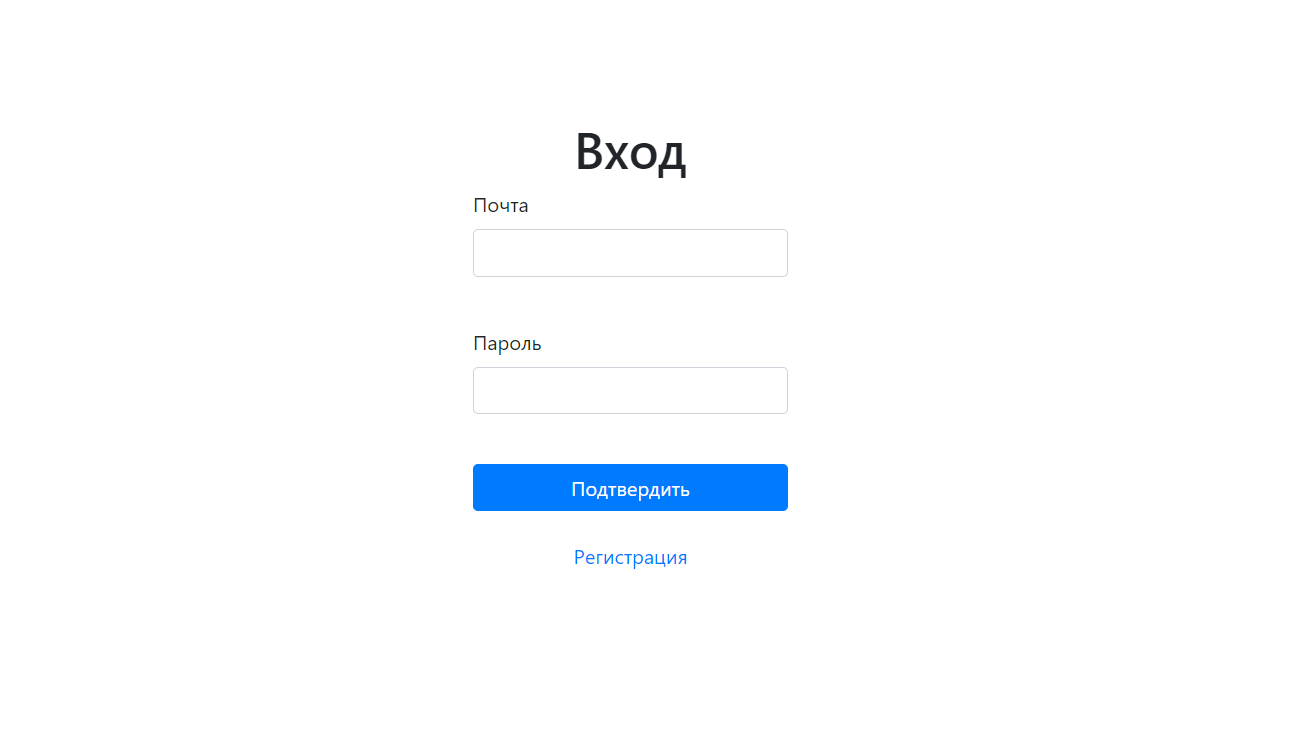


Рисунок 8.1 – Страница авторизации

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

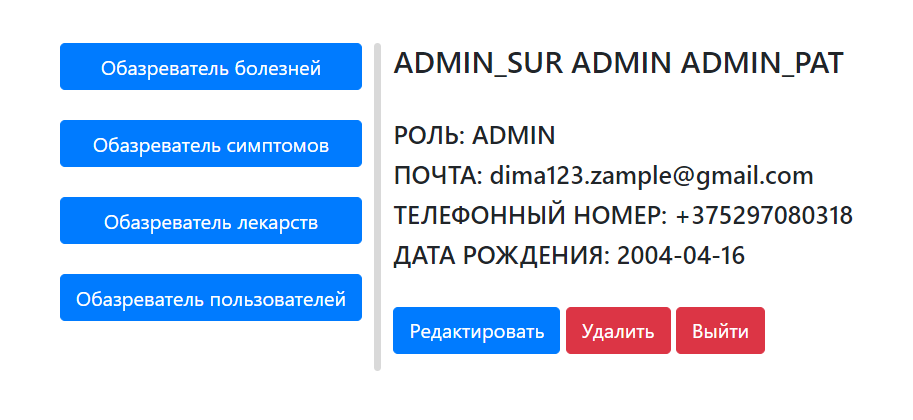


Рисунок 8.2 – Главная страница админа

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

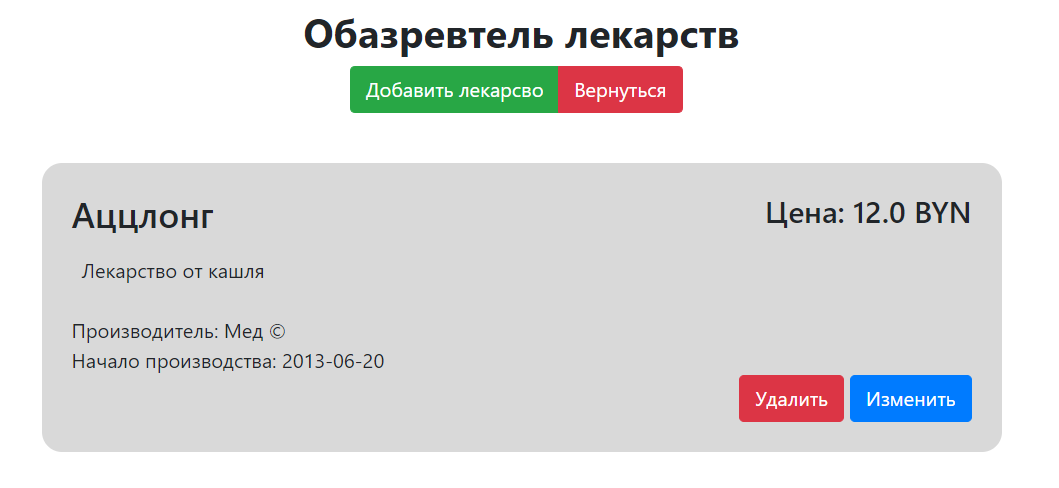


Рисунок 8.3 – Обозреватель лекарств

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.

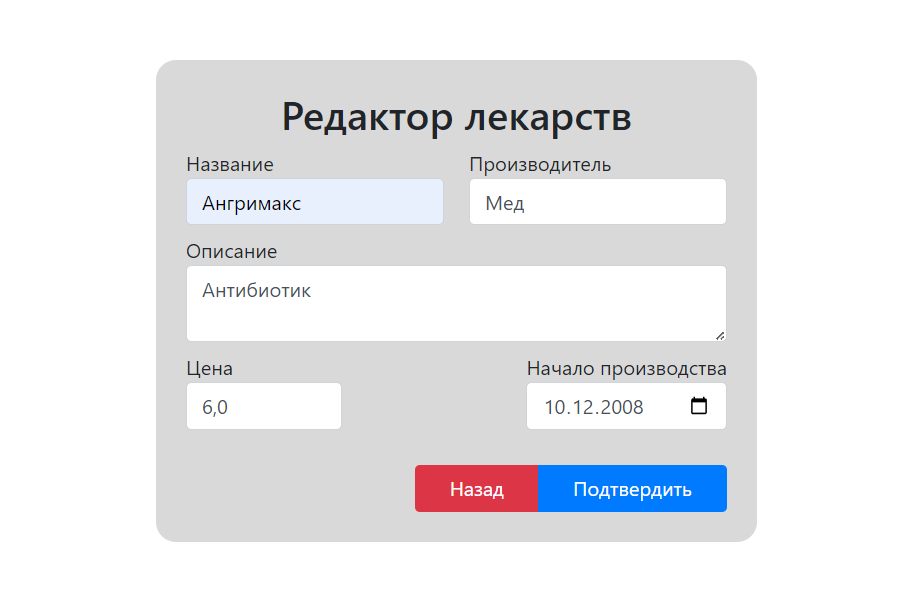


Рисунок 8.4 – Редактор лекарств

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.

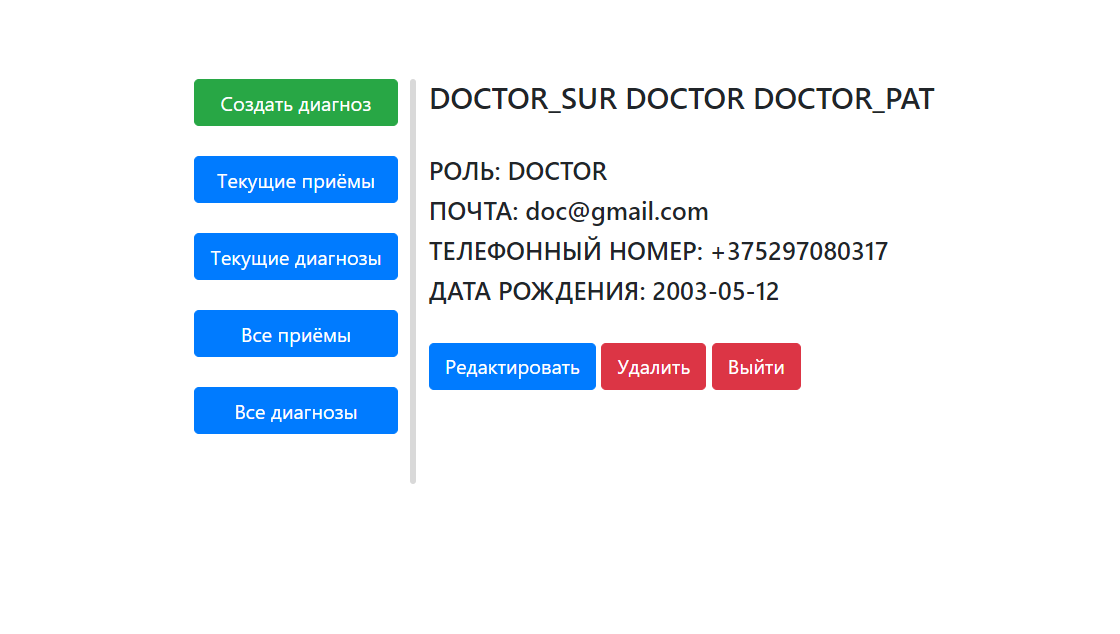


Рисунок 8.5 – Главная страница врача

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;