МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Базы данных»

Тема «Реализация базы данных медицинского центра с использованием технологии шифрования и маскирования БД»

**Исполнитель**

студент 2 курса 7 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. А. Воликов

подпись, дата

**Руководитель**

Ассистент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кантарович В. С.

должность, учен. степень, ученое звание подпись, дата

Допущен к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_ Кантарович В. С.

подпись дата инициалы и фамилия

**Содержание**

[Введение 6](#_Toc135270869)

[1. Анализ требований к программному средству 7](#_Toc135270870)

[1.1. Аналитический обзор аналогов 7](#_Toc135270871)

[1.1.1. Аналог «Talon.by» 7](#_Toc135270872)

[1.1.2. Аналог «Городская студенческая поликлиника» 7](#_Toc135270873)

[1.1.3. Аналог «3-я центральная районная клиническая поликлиника» 8](#_Toc135270874)

[1.2. Разработка функциональных требований, определение вариантов использования 9](#_Toc135270875)

[1.2.1. Разработка требований к базе данных 9](#_Toc135270876)

[1.2.2. Определение вариантов использования 9](#_Toc135270877)

[1.3. Вывод 10](#_Toc135270878)

[2. Разработка архитектуры проекта 11](#_Toc135270879)

[2.1. Обобщенная структура управления приложением 11](#_Toc135270880)

[2.2. Диаграммы UML, взаимосвязь их компонентов 12](#_Toc135270881)

[2.3. Описание информационных объектов и ограничений целостности 13](#_Toc135270882)

[2.3.1 Таблица ROLES 14](#_Toc135270883)

[2.3.2 Таблица USERS 14](#_Toc135270884)

[2.3.3 Таблица PERSONS 14](#_Toc135270885)

[2.3.4 Таблица PATIENTS 15](#_Toc135270886)

[2.3.5 Таблица EMPLOYEES 15](#_Toc135270887)

[2.3.6 Таблица TALONS 16](#_Toc135270888)

[2.3.7 Таблица TREATMENTS 16](#_Toc135270889)

[2.3.8 Таблица ADDRESSES 16](#_Toc135270890)

[2.3.9 Таблица PERSON\_ADDRESS 17](#_Toc135270891)

[2.3.10 Таблица PASSPORTS 17](#_Toc135270892)

[2.3.11 Таблица DEAPARTMENTS 17](#_Toc135270893)

[2.3.12 Таблица DEPARTMENT\_EMPLOYEE 18](#_Toc135270894)

[2.3.13 Таблица POSITIONS 18](#_Toc135270895)

[2.3.14 Таблица PRICELIST 18](#_Toc135270896)

[2.3.15 Таблица SUPPLIERS 19](#_Toc135270897)

[2.3.16 Таблица PHARMACY 19](#_Toc135270898)

[2.3.17 Таблица COMMENTS 19](#_Toc135270899)

[2.4. Вывод 20](#_Toc135270900)

[3. Разработка модели базы данных 21](#_Toc135270901)

[3.1. Создание необходимых объектов 21](#_Toc135270902)

[3.1.1 Табличные пространства 21](#_Toc135270903)

[3.1.2. Таблицы 21](#_Toc135270904)

[3.1.3. Пакеты и хранимые процедуры 22](#_Toc135270905)

[3.1.4 Индексы 24](#_Toc135270906)

[3.1.5 Функции 24](#_Toc135270907)

[3.1.6. Триггеры 25](#_Toc135270908)

[3.2. Описание используемой технологии 25](#_Toc135270909)

[3.3. Вывод 27](#_Toc135270910)

[4. Установка, настройка и использование Oracle 19c 28](#_Toc135270911)

[4.1. Создание ролей для разграничения доступа 28](#_Toc135270912)

[4.2. Описание процедур экспорта и импорта данных 31](#_Toc135270913)

[4.3. Заполнение таблицы 100000 строк 32](#_Toc135270914)

[4.4. Тестирование производительности базы данных 34](#_Toc135270915)

[4.5. Вывод 35](#_Toc135270916)

[5. Тестирование, проверка работоспособности и анализ полученных данных 36](#_Toc135270917)

[5.1. Тестирование клиентской части 36](#_Toc135270918)

[5.2. Тестирование фармацевтической части 37](#_Toc135270919)

[5.3. Тестирование врачебной части 38](#_Toc135270920)

[5.4. Тестирование части менеджера 39](#_Toc135270921)

[5.5. Проверка работоспособности процедур импорта и экспорта 40](#_Toc135270922)

[5.6. Вывод 42](#_Toc135270923)

[6. Руководство по использованию программного средства 43](#_Toc135270924)

[6.1. Установка приложения 43](#_Toc135270925)

[6.2. Панель менеджера 44](#_Toc135270926)

[6.3. Клиентская часть приложения 45](#_Toc135270927)

[6.3. Вывод 48](#_Toc135270928)

[Заключение 49](#_Toc135270929)

[Список использованных литературных источников 50](#_Toc135270930)

[Приложение А 51](#_Toc135270931)

[Приложение Б 57](#_Toc135270932)

# Введение

Современная разработка приложений, которые нацелены на долгосрочную работу с пользователем, невозможно представить без какой-либо базы данных. Базы данных хранят в себе информацию, которую требуют сервисы для своей стабильной и удобной для пользователя работы.

Сегодня в сети можно найти множество различных данных, которые можно использовать с различной целью. И так, как все приложения хранят конфиденциальную информацию о пользователях, из этого следует то, что весь поток личных данных требуется шифровать для защиты от просмотра и использования 3-ми лицами.

Для курсового проекта было решено спроектировать и разработать базу данных для медицинского центра. Медицинский центр – отличный пример, где требуется некого рода хранилище данных: о пациентах (личная информация, диагнозы и т.п.), о сотрудниках (дата найма, должность, уровень зарплаты и т.п.). А так, как там хранится некие личные данные, которые должны быть известны только медицинскому центру и пациентам, то потребуется настроить шифрование данных.

Цель курсового проекта – разработка и реализация базы данных Oracle и интерфейса для неё.

К задачам курсового проекта относится: аналитический обзор литературы по теме проекта, изучение требований; определение вариантов использования; анализ и проектирование модели данных; описание информационных объектов и ограничений целостности; создание необходимых объектов; импорт и экспорт данных; описание требуемой технологии; тестирование производительности; формирование вывода по каждому разделу; заключение, включающее вывод по проделанной работе.

# 1. Анализ требований к программному средству

## 1.1. Аналитический обзор аналогов

Любой современный медицинский центр обязан иметь своё собственное приложение. Технологии сегодня позволяют любому человеку забронировать талон к врачу, не выходя из дома (особенно в период пандемии). Ключевым фактором проектирования приложения для медицинского центра является удобство.

Будем честны – на сегодняшний день очень много сайтов с тематикой медицинского центра. Однако все они оставляют желать лучшего. Можно даже добавить то, что почти никто не пользуется сайтом поликлиник (только в случае надобности номера телефона).

1.1.1. Аналог «Talon.by»

Пожалуй, единственный удобный сайт, где можно заказать талон к любому врачу из любой поликлиники – это «Talon.by» [1]. Данный сервис довольно удобен в использовании: пользователь легко может просмотреть свои забронированные талон (в случае чего, удалить бронь). Также этот сервис реализовал довольно удобную функцию создания нескольких пациентов на одного пользователя, что может быть удобно для некоторых случаев (родитель и дети, пользователь и пожилой родственник и т.п.).

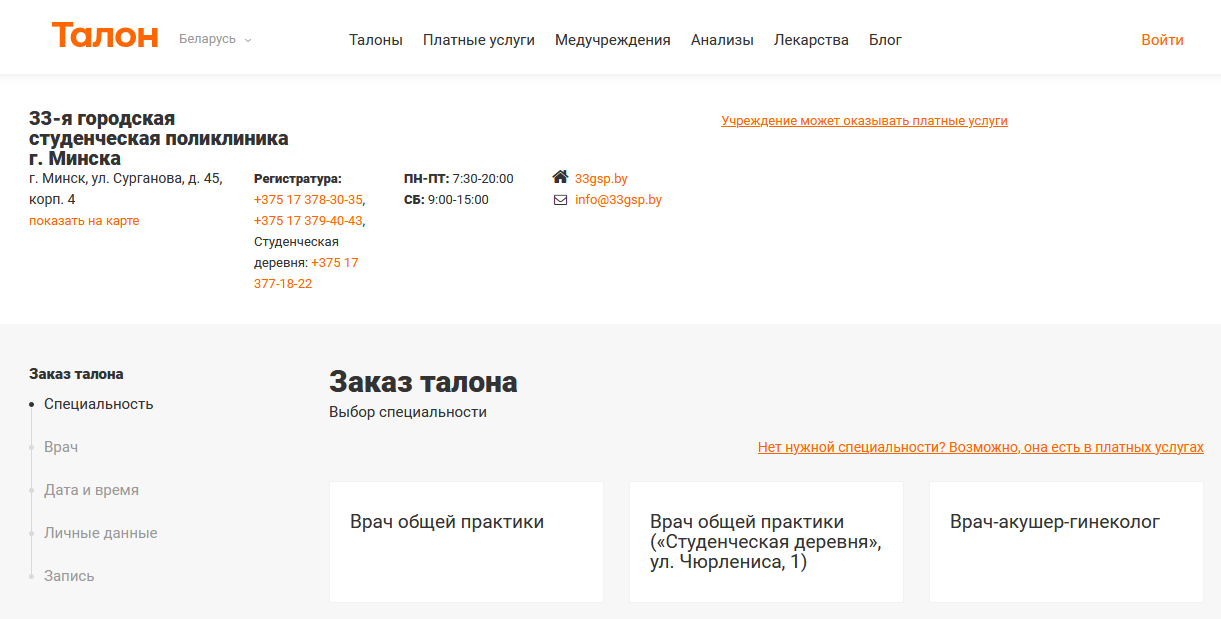


Рисунок 1.1 – Страница заказа талона для определённой поликлиники

У данного сайта можно выделить хорошее разделение по типам должностей врачей, где уже потом можно найти все доступные талоны к этим врачам.

1.1.2. Аналог «Городская студенческая поликлиника»

«Городская студенческая поликлиника» [2] – данный сайт предоставляет по большей части лишь только ознакомительную информацию, которая будет полезна пользователю: список цен платных услуг, контактные данные, адрес поликлиники и т.п. Однако лишь только ознакомительную, а значит много с этим сайтом не сделаешь. Хотя можно похвалить сайт за некое единство стилей.

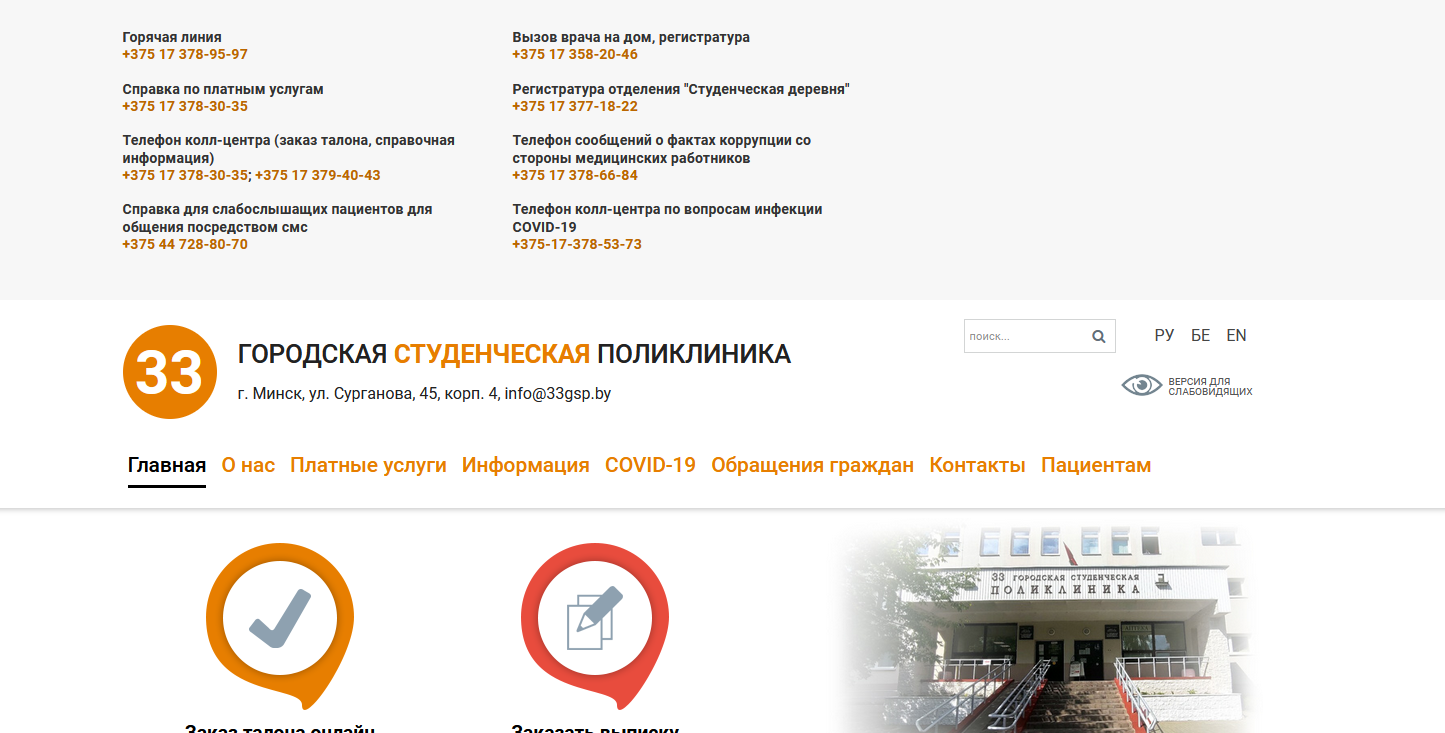


Рисунок 1.2 – Страница сайта студенческой поликлиники

Данный сайт не имеет своей системы регистрации талонов. Чтобы заказать талон к врачу, требуется перейти на другой источник – Talon.by, который был рассмотрен выше.

1.1.3. Аналог «3-я центральная районная клиническая поликлиника»

«3-я центральная районная клиническая поликлиника» [3] – данный сайт, как и предыдущий аналог, разрабатывала одна фирма. Минимальные различия лишь только в дизайне компонентов и в цветах. Однако, как и в прошлом аналоге, стоит отметить единство стиля: хорошо подобранный цвет, центрирование и расположение элементов, боковые отступы.

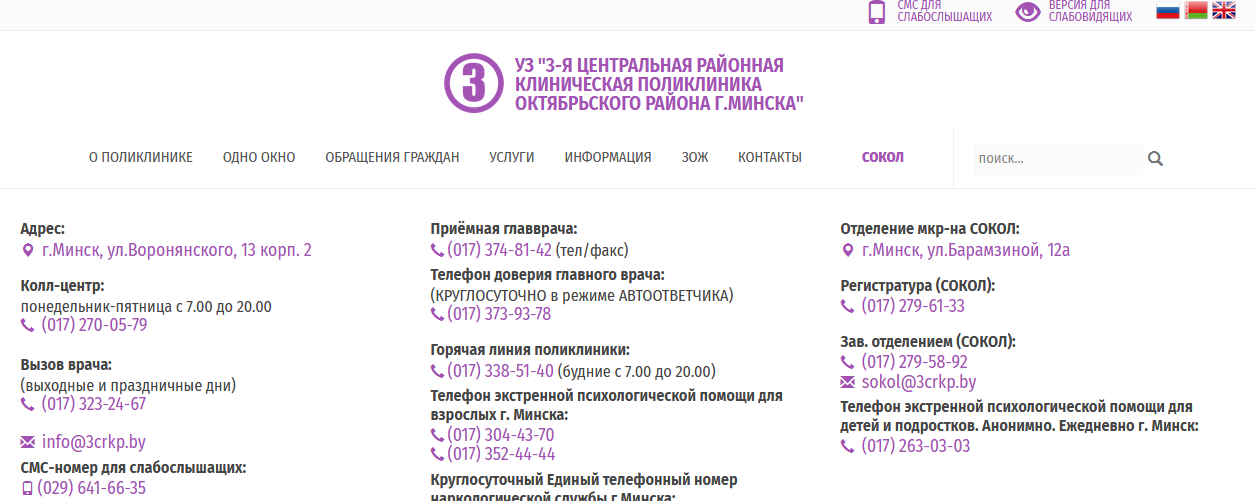


Рисунок 1.3 – Главная страница 3-ей поликлиники

Данный сайт, как и прошлый, не имеет своей системы регистрации талонов. Он также ссылается на сайт Talon.by.

## 1.2. Разработка функциональных требований, определение вариантов использования

Важным этапом разработки проекта является изучение требований и определение возможных вариантов использования. Требования к проекту могут быть как функциональными, то есть связанными с основными функциями проекта, так и нефункциональными, то есть связанными с качественными характеристиками проекта.

1.2.1. Разработка требований к базе данных

Медицинский центр – это довольно комплексный проект, который состоит из большого количества различных частей. Здесь можно выделить следующие этапы, которые будут присущи базе данных проекта:

1. Хранение информации о пользователях (данные авторизации), о пациентах и сотрудниках, которые привязаны к пользователю (личностные характеристики, номер телефона, адрес(-а) и т.д.).
2. Создание нескольких пациентов на одного пользователя, то есть один пользователь может иметь несколько данных о некоторых людях, на которых будет бронироваться талон.
3. Предоставление информации о талонах, об отделениях и их расположения, о сотрудниках и их специальности.
4. Разработка приложения для взаимодействия с базой данных.
5. Создание основных ролей базы данных (пользователь, менеджер)
6. Обеспечение защиты критически важных данных (пароли пользователей)
7. База данных должна иметь высокую производительность и обеспечивать быстрый доступ к данным для обеспечения эффективной работы медицинского центра.
8. База данных должна быть стабильной, обеспечивать целостность данных и иметь возможность быстрого восстановления после сбоев.
9. База данных должна быть удобной и простой в использовании для обеспечения эффективной работы пользователей.

1.2.2. Определение вариантов использования

Основными вариантами использования можно выделить следующие пункты:

* Авторизация и регистрация пользователей.
* Добавление пациентов у пользователя.
* Редактирование информации о пациентах пользователя.
* Просмотр доступных талонов.
* Бронирование талона для пациента.
* Администрирование со стороны менеджера.

## 1.3. Вывод

Аналитический анализ дал некое понимание того, как следует реализовать базу данных и различные объекты. Например, так, как различных талонов хранится в базе данных достаточно много, то следует создать индекс на таблицу талонов.

Обзор аналогов позволил определить структуру клиентского приложения, основной функционал и требования к реализации. Например, пользователь может войти в аккаунт (или зарегистрироваться, если аккаунт отсутствует), создать пациента и заказать талон на данного пациента. В свою очередь, менеджер может управлять всеми данными, что представлены в базе данных.

# 2. Разработка архитектуры проекта

## 2.1. Обобщенная структура управления приложением

Пользовательский интерфейс взаимодействует с бизнес-логикой, которая в свою очередь взаимодействует со слоем доступа к данным. Слой доступа к данным выполняет запросы к базе данных Oracle и возвращает результаты обратно в бизнес-логику, которая обрабатывает эти результаты и возвращает их обратно в пользовательский интерфейс. База данных содержит данные, с которыми приложение работает.

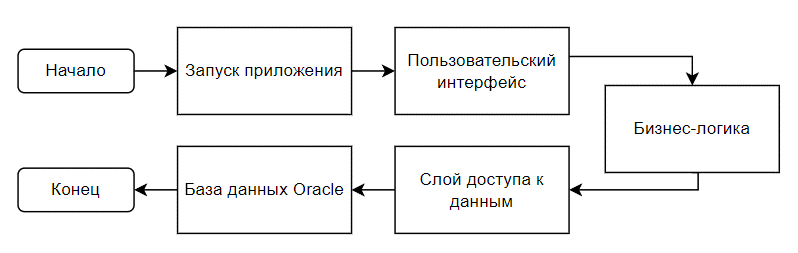


Рисунок 2.1 – Обобщенная структура управления приложением

Для разработки приложения был выбран популярный PHP-фреймворк Yii2, который реализует паттерн MVC(Model-View-Controller). Это довольно мощный фреймворк для быстрого создания веб-приложений. Yii2 поддерживает многие современные технологии, такие как RESTful API, кеширование, тестирование, интернационализация и другие. Yii2 также имеет мощный инструмент для генерации кода, называемый Gii, который помогает создавать модели, контроллеры, представления и CRUD операции.

Yii2 фреймворк – это PHP-фреймворк, а значит требуется некий веб сервер. Nginx - это веб-сервер и обратный прокси-сервер, который может также работать как почтовый прокси-сервер и TCP/UDP прокси-сервер. nginx был изначально написан Игорем Сысоевым и используется на многих популярных сайтах, таких как Yandex, Mail.Ru, VK и Rambler.

Соединение с базой данных происходит довольно легко за счёт расширения языка PHP Data Objects (PDO), который определяет легкий, согласованный интерфейс для доступа к базам данных в PHP. Каждый драйвер базы данных, реализующий интерфейс PDO, может предоставлять функции, специфичные для базы данных, как обычные функции расширения.

Фреймворк настраивает связь с базой данных, получает информацию из таблиц и выводит всё в интерфейс пользователя. После чего, связь обрывается, что позволяет освободить ресурсы системы.

## 2.2. Диаграммы UML, взаимосвязь их компонентов

Диаграмма UML позволяет нам визуализировать связь и отношения всех компонентов в системы базы данных, а также описать функциональность системы и её взаимодействие с внешними пользователями или другими системами.

В ходе этапа анализа требований к программному средству были выделены основные две роли: пользователь и менеджер. К данными ролям добавляются ещё две роли, которые реализует поведение сотрудника медицинского центра – врач и фармацевт.

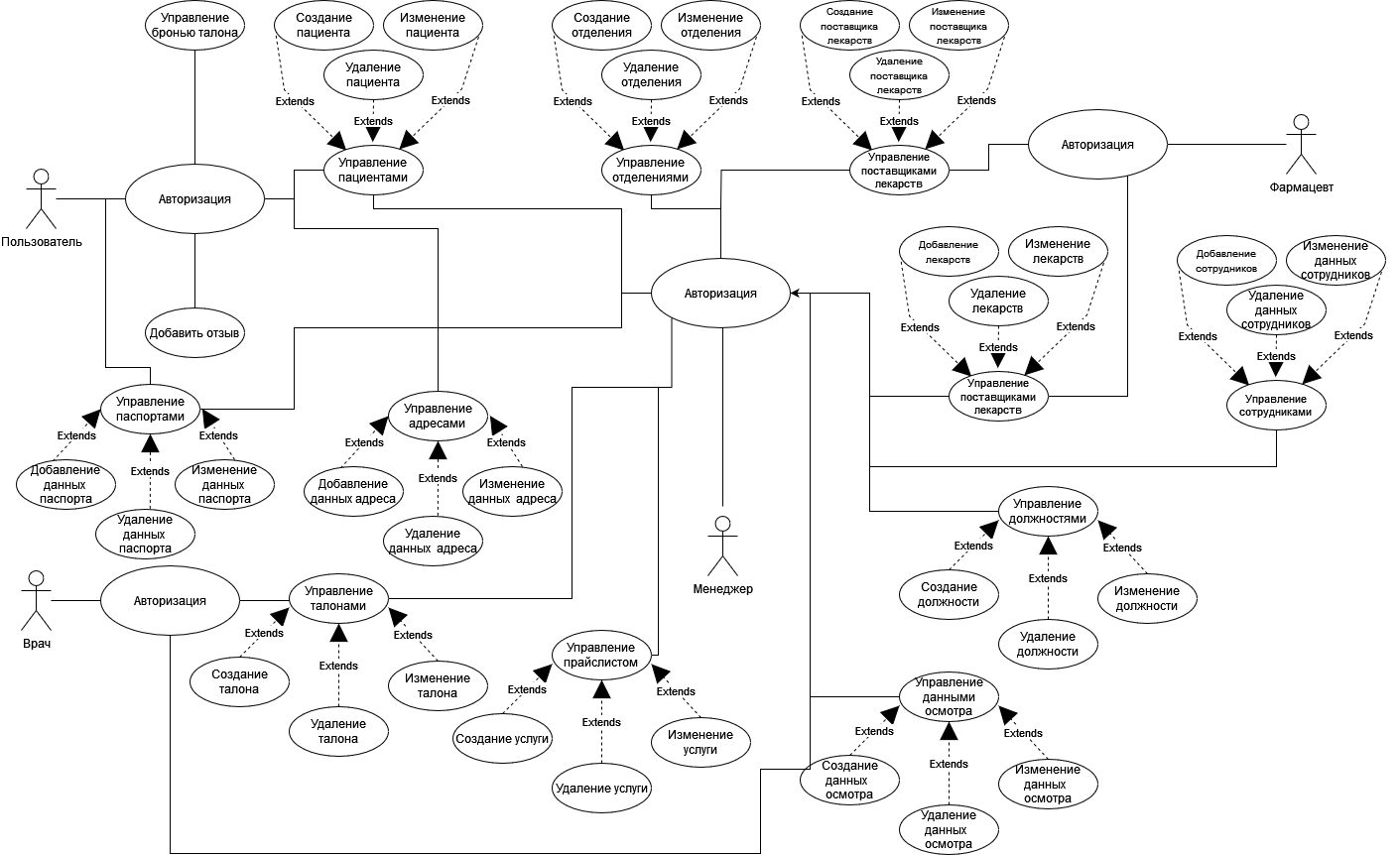


Рисунок 2.2 – Диаграмма вариантов использования

По схеме видно, что роль менеджера по иерархии выше остальных, ведь ему доступны абсолютно любое действие и абсолютно любая информация. Следующие две роли – врач и фармацевт – их возможности ограничены только их же задачами. Фармацевт – управляет аптекой медицинского центра: регистрация поставщиков лекарственных препаратов и самих же лекарственных препаратов. Врач – это пользователь, который проводит осмотры пациентам и ставит им диагноз. Пользователь – это тот, кто имеет возможность брони талона к врачу, который был описан выше. Также пользователь имеет возможность добавить несколько пациентов на свой аккаунт, чтобы при бронировании талона указать данные о пациенте.

2.3. Описание информационных объектов и ограничений целостности

В ходе анализа функциональных требований был реализован ряд таблиц, каждая из которых играет важную роль в хранении данных. Структуру таблиц, их связи и ограничения можно наблюдать на изображении ниже.

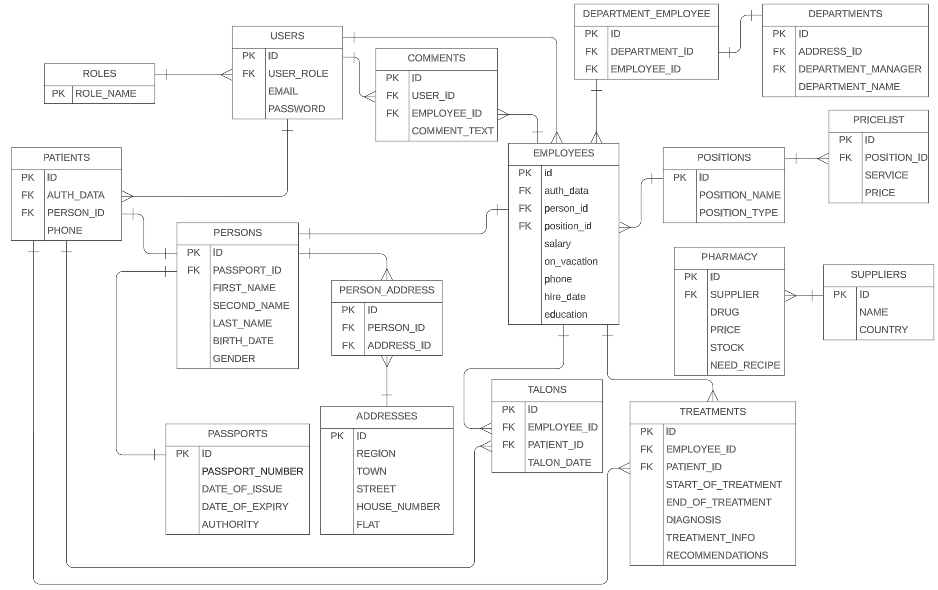


Рисунок 2.3 – Структурная диаграмма базы данных

Проектирование таблиц было выполнено с учётом первых трёх правил нормализации, чтобы избежать некой избыточности некоторой информации. Было создано 17 таблиц.

Основные таблицы:

* USERS. Данные пользователей.
* PERSONS. Личностные данные пользователей.
* PATIENTS. Данные пациентов.
* EMPLOYEES. Данные сотрудников.
* DEPARTMENTS. Данные отделений поликлиники.
* TALONS. Данные талонов к врачам.
* PHARMACY. Данные об аптеке медицинского центра.
* TREATMENTS. Данные об осмотрах пациентов у врача.

Данные таблицы можно считать основными, так как без них невозможна даже самая минимальная реализация медицинского центра, у которой есть возможность регистрации пользователей, онлайн бронирования талонов и редактирования данных о пациенте пользователя.

Дополнительные таблицы:

* ADDRESSES. Данные адресов сотрудников, пациентов и отделений.
* PASSPORTS. Данные пациентов и сотрудников.
* ROLES. Данные ролей пользователей.
* SUPPLIERS. Данные поставщиков лекарственных препаратов.
* PERSON\_ADDRESS. Таблица связывания пациента или сотрудника с адресом проживания. Данная таблица реализует связь многие ко многим, так, как один пациент или пользователь может иметь несколько адресов проживания.
* PRICELIST. Данные о платных услугах медицинского центра.
* POSITIONS. Данные о должностях сотрудников.
* DEPATMENT\_EMPLOYEE. Таблица связывания сотрудника с отделением медицинского центра.
* COMMENTS. Данные об отзывах пользователей о сотрудниках.

2.3.1 Таблица ROLES

Данная таблица используется для хранения возможных ролей.

В состав таблицы входит один столбец – ROLE\_NAME. Хранит имя роли, которое будет выдаваться пользователям. Данный столбец имеет ограничение первичного ключа.

2.3.2 Таблица USERS

Данная таблица используется для хранения данных авторизации пользователей.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого пользователя.
* Столбец USER\_ROLE. Хранит роль пользователя.
* Столбец EMAIL. Хранит адрес электронной почты пользователя
* Столбец PASSEORD. Хранит пароль пользователя.

Данная таблица содержит ограничение первичного ключа для столбца ID, ограничение внешнего ключа USER\_ROLE, ограничение на уникальность EMAIL, все столбцы обязаны иметь значения, отличные от null.

2.3.3 Таблица PERSONS

Данная таблица используется для хранения личностных данных сотрудников и пациентов.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждой записи.
* Столбец PASSPORT\_ID. Хранит идентификатор на запись паспорта в таблице PASSPORTS.
* Столбец FIRST\_NAME. Хранит значение имени пациента (сотрудника).
* Столбец SECOND\_NAME. Хранит значение фамилии пациента (сотрудника).
* Столбец LAST\_NAME. Хранит значение отчества пациента (сотрудника) (обычно для граждан СНГ).
* Столбец GENDER. Хранит значение пола пациента (сотрудника).
* Столбец BIRTH\_DATE. Хранит значение даты рождения пациента (сотрудника).

Данная таблица содержит ограничение первичного ключа для столбца ID, ограничения внешнего ключа и уникальности PASSPORT\_ID, все столбцы обязаны иметь значения, отличные от null, кроме PASSPORT\_ID.

2.3.4 Таблица PATIENTS

Данная таблица используется для хранения данных пациентов пользователя.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого пациента.
* Столбец PERSON\_ID. Хранит идентификатор на личностные данные из таблицы PERSONS.
* Столбец AUTH\_DATA. Хранит идентификатор на пользователя из таблицы USERS.
* Столбец PHONE. Хранит значение номера телефона пациента.

Данная таблица содержит ограничение первичного ключа для столбца ID, ограничения внешнего ключа на PERSON\_ID и AUTH\_DATA, все столбцы обязаны иметь значения, отличные от null.

2.3.5 Таблица EMPLOYEES

Данная таблица используется для хранения данных сотрудников.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого сотрудника.
* Столбец PERSON\_ID. Хранит идентификатор на личностные данные из таблицы PERSONS.
* Столбец AUTH\_DATA. Хранит идентификатор на пользователя из таблицы USERS.
* Столбец POSITION\_ID. Хранит идентификатор на должность из таблицы POSITIONS.
* Столбец PHONE. Хранит значение номера телефона сотрудника.
* Столбец HIRE\_DATE. Хранит значение даты найма.
* Столбец SALARY. Хранит значение заработной платы сотрудника.
* Столбец EDUCATION. Хранит значение образования сотрудника.
* Столбец ON\_VACATION. Хранит значения того, находится ли сотрудник в отпуске.

Данная таблица содержит ограничение первичного ключа для столбца ID, ограничения внешнего ключа и уникальности на PERSON\_ID и AUTH\_DATA, ограничение внешнего ключа для POSITION\_ID, все столбцы обязаны иметь значения, отличные от null.

2.3.6 Таблица TALONS

Данная таблица используется для хранения данных о талонах.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого талона.
* Столбец EMPLOYEE\_ID. Хранит идентификатор на сотрудника из таблицы PERSONS.
* Столбец PATIENT\_ID. Хранит идентификатор на пациента из таблицы PATIENTS.
* Столбец TALON\_DATE. Хранит значение даты приёма талона.

Данная таблица содержит ограничение первичного ключа для столбца ID, ограничения внешнего ключа на EMPLOYEE\_ID и PATIENT\_ID, все столбцы обязаны иметь значения, отличные от null, кроме столбца PATIENT\_ID.

2.3.7 Таблица TREATMENTS

Данная таблица используется для хранения данных о планах лечения пациентов.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждой записи осмотра.
* Столбец EMPLOYEE\_ID. Хранит идентификатор на сотрудника из таблицы PERSONS.
* Столбец PATIENT\_ID. Хранит идентификатор на пациента из таблицы PATIENTS.
* Столбец START\_OF\_TREATMENT. Хранит значение даты приёма (лечения).
* Столбец END\_OF\_TREATMENT. Хранит значение даты окончания лечения.
* Столбец DIAGNOSIS. Хранит значение диагноза.
* Столбец TREATMENT\_INFO. Хранит информацию о ходе осмотра и лечения.
* Столбец RECOMMENDATIONS. Хранит информацию о рекомендациях от врача.

Данная таблица содержит ограничение первичного ключа для столбца ID, ограничения внешнего ключа на EMPLOYEE\_ID и PATIENT\_ID, все столбцы обязаны иметь значения, отличные от null, кроме столбца END\_OF\_TREATMENT.

2.3.8 Таблица ADDRESSES

Данная таблица используется для хранения данных об адресах пациентов и сотрудников. Требуется для того, чтобы реализовать связь многие ко многим между пациентами и сотрудниками с адресами проживания.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого адреса.
* Столбец REGION. Хранит значение области.
* Столбец TOWN. Хранит значение населённого пункта.
* Столбец STREET. Хранит значение улицы дома.
* Столбец HOUSE\_NUMBER. Хранит значение номера дома.
* Столбец FLAT. Хранит значение номера квартиры (комнаты).

Данная таблица содержит ограничение первичного ключа для столбца ID, все столбцы обязаны иметь значения, отличные от null, кроме столбца FLAT.

2.3.9 Таблица PERSON\_ADDRESS

Данная таблица используется для хранения связи пациента (сотрудника) с адресом (-ами).

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждой записи.
* Столбец PERSON\_ID. Хранит идентификатор на личностные данные пациента (сотрудника) из таблицы PERSONS.
* Столбец ADDRESS\_ID. Хранит идентификатор на адрес из таблицы ADDRESSES.

Данная таблица содержит ограничение первичного ключа для столбца ID, ограничения внешнего ключа на ADDRESS\_ID и PERSON\_ID, все столбцы обязаны иметь значения, отличные от null.

2.3.10 Таблица PASSPORTS

Данная таблица используется для хранения данных о паспорте.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого паспорта.
* Столбец PASSPORT\_NUMBER. Хранит номер паспорта.
* Столбец DATE\_OF\_ISSUE. Хранит дату выдачи паспорта.
* Столбец DATE\_OF\_EXPIRY. Хранит дату изъятия паспорта.
* Столбец AUTHORITY. Хранит значения представителей власти, которые выдали паспорт.

Данная таблица содержит ограничение первичного ключа для столбца ID, ограничения уникальности на PASSPORT\_NUMBER, все столбцы обязаны иметь значения, отличные от null.

2.3.11 Таблица DEAPARTMENTS

Данная таблица используется для хранения данных об отделениях медицинского центра.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого отделения.
* Столбец DEPARTMENT\_NAME. Хранит значение названия отделения.
* Столбец DEPARTMENT\_MANAGER. Хранит идентификатор на сотрудника из таблиц EMPLOYEES.
* Столбец ADDRESS\_ID. Хранит идентификатор на адрес из таблицы ADDRESSES.

Данная таблица содержит ограничение первичного ключа для столбца ID, ограничения внешнего ключа на ADDRESS\_ID и DEPARTMENT\_MANAGER, все столбцы обязаны иметь значения, отличные от null, кроме DEPARTMENT\_MANAGER.

2.3.12 Таблица DEPARTMENT\_EMPLOYEE

Данная таблица используется для хранения связи сотрудника с отделением.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждой записи.
* Столбец EMPLOYEE\_ID. Хранит идентификатор на сотрудника из таблицы EMPLOYEES.
* Столбец DEPARTMENT\_ID. Хранит идентификатор на отделение из таблицы DEPARTMENTS.

Данная таблица содержит ограничение первичного ключа для столбца ID, ограничения внешнего ключа на EMPLOYEE\_ID и DEPARTMENT\_ID, все столбцы обязаны иметь значения, отличные от null.

2.3.13 Таблица POSITIONS

Данная таблица используется для хранения данных о должностях.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждой должности.
* Столбец POSITION\_NAME. Хранит значения названия должности.
* Столбец POSITION\_TYPE. Хранит общее название для сотрудников со схожей должностью.

Данная таблица содержит ограничение первичного ключа для столбца ID, все столбцы обязаны иметь значения, отличные от null.

2.3.14 Таблица PRICELIST

Данная таблица используется для хранения данных о платных услугах.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждой услуги.
* Столбец POSITION\_ID. Хранит идентификатор на должность из таблицы POSITIONS.
* Столбец SERVICE. Хранит значение названия услуги.
* Столбец PRICE. Хранит значение цены услуги.

Данная таблица содержит ограничение первичного ключа для столбца ID, ограничение внешнего ключа на POSITION\_ID, все столбцы обязаны иметь значения, отличные от null.

2.3.15 Таблица SUPPLIERS

Данная таблица используется для хранения данных о поставщиках лекарственных препаратов.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого поставщика.
* Столбец NAME. Хранит значение названия поставщика.
* Столбец COUNTRY. Хранит значение страны поставщика.

Данная таблица содержит ограничение первичного ключа для столбца ID, все столбцы обязаны иметь значения, отличные от null.

2.3.16 Таблица PHARMACY

Данная таблица используется для хранения данных об аптеке.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждой лекарственного препарата.
* Столбец SUPPLIER. Хранит идентификатор на поставщика из таблицы SUPPLERS.
* Столбец DRUG. Хранит значение название лекарственного препарата.
* Столбец PRICE. Хранит значение цены лекарственного препарата.
* Столбец STOCK. Хранит значение количества лекарственных препаратов.
* Столбец NEED\_RECIPE. Хранит значение того, нужен ли рецепт для лекарственного препарата.

Данная таблица содержит ограничение первичного ключа для столбца ID, ограничение внешнего ключа на SUPPLIER\_ID, все столбцы обязаны иметь значения, отличные от null.

2.3.17 Таблица COMMENTS

Данная таблица используется для хранения отзывов о сотрудниках от пользователей.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого отзыва.
* Столбец EMPLOYEE\_ID. Хранит идентификатор на сотрудника из таблицы EMPLOYEES.
* Столбец USER\_ID. Хранит идентификатор на пользователя из таблицы USERS.

Данная таблица содержит ограничение первичного ключа для столбца ID, ограничения внешнего ключа на EMPLOYEE\_ID и USERS\_ID, все столбцы обязаны иметь значения, отличные от null.

## 2.4. Вывод

В ходе этого этапы были спроектированы все необходимы таблицы, которые будут хранить все возможные данные о медицинском центре, и непосредственно с которыми будут работать выше перечисленные роли: менеджер, врач, фармацевт и пользователь. Каждый пользователь базы данных работает только с теми таблицами и данными, на который он имеет права доступа.

Любая база данных должна быть спроектирована по правилам нормализации. Они придуманы для того, чтобы избежать ошибок при работе с продуктом: избыточности информации, отсутствия атомарности объектов, некорректная зависимость столбцов таблиц с первичными ключами и т.п. Наличие таких ошибок говорит о том, что база данных спроектирована неверно и требуется некоторые изменения в структуре таблиц.

# 3. Разработка модели базы данных

## 3.1. Создание необходимых объектов

Для реализации базы данных медицинского центра необходимо создать набор объектов базы данных, которые представляют собой табличные пространства, таблицы, индексы, функции, триггеры, пакеты и хранимые процедуры.

3.1.1 Табличные пространства

Табличное пространство – это логическая структура, которая используется для группировки данных с однотипными методами доступа. В табличное пространство может входить, один или несколько файлов данных. Но, как правило, один или несколько файлов данных не могут хранить два табличных пространства, а только одно. Пример создания табличного пространства приведён в листинге 3.1.

CREATE TABLESPACE TS\_USERS

DATAFILE 'TS\_USERS.dbf'

SIZE 50M

AUTOEXTEND ON NEXT 10M

EXTENT MANAGEMENT LOCAL;

Листинг 3.1 – SQL код создания табличного пространства

Табличные пространства, разработанные в рамках курсового проекта:

* TS\_USERS. Используется для хранения данных, которые относятся ко всем возможным пользователям медицинского центра.
* TS\_MEDCENTER. Используется для хранения данных, которые относятся непосредственное к медицинскому центру.

3.1.2. Таблицы

Таблица – это логическая сущность, которая делает чтение и манипуляции данных интуитивно понятными для пользователя. Таблица состоит из столбцов и строк, причем строка соответствует одиночной записи, которая состоит из набора полей. Пример создания таблицы приведён в листинге 3.2.

CREATE TABLE ROLES

(

role\_name NVARCHAR2(64) PRIMARY KEY

) TABLESPACE TS\_USERS;

Листинг 3.2 – SQL код создания таблицы

Все таблицы, которые были созданы в рамках курсового проекта были описаны в предыдущем разделе.

3.1.3. Пакеты и хранимые процедуры

Процедура представляет собой модуль, выполняющий одно или несколько действий. Поскольку вызов процедуры в PL/SQL является отдельным исполняемым оператором, блок кода PL/SQL может состоять только из вызова процедуры.

Пакет Oracle PL/SQL - это объект схемы, который группирует логически связанные типы, элементы и подпрограммы. Пакеты обычно состоят из двух частей: спецификации и тела, хотя иногда тело не нужно. Спецификация - это интерфейс для приложений. В спецификации пакета объявляются типы, переменные, константы, исключения, курсоры и подпрограммы, доступные для использования.  
Тело пакета полностью определяет курсоры и подпрограммы и реализует спецификацию.

Пример создания пакета приведён в листинге 3.3.

create or replace package DEPARTMENTS\_tapi

is

PROCEDURE create\_department

(

p\_dep\_name in DEPARTMENTS.department\_name%TYPE,

p\_dep\_manager in DEPARTMENTS.department\_manager%TYPE

);

PROCEDURE update\_department

(

p\_id in DEPARTMENTS.id%TYPE,

p\_dep\_name in DEPARTMENTS.department\_name%TYPE,

p\_dep\_manager in DEPARTMENTS.department\_manager%TYPE

);

PROCEDURE delete\_department

(

p\_id in DEPARTMENTS.id%TYPE

);

END DEPARTMENTS\_tapi;

Листинг 3.3 – Спецификация пакета

Пример создания процедуры приведён в листинге 3.4.

PROCEDURE delete\_department

(p\_id in DEPARTMENTS.id%TYPE

)IS BEGINDELETE FROM DEPARTMENTS WHERE id = p\_id;

COMMIT;END;

Листинг 3.4 – Процедура удаления отедления

Пакеты и процедуры, которые были созданы в рамках курсового проекта:

* USERS\_tapi. Пакет, который хранит процедуры create\_user (создание пользователя), update\_user (редактирование пользователя) и delete\_user (удаление пользователя).
* ADDRESSES\_tapi. Пакет, который хранит процедуры create\_address (создание адреса), update\_address (редактирование адреса) и delete\_address (удаление адреса).
* PASSPORTS\_tapi. Пакет, который хранит процедуры create\_passport (создание паспорта), update\_passport (редактирование паспорта) и delete\_passport (удаление паспорта).
* PATIENTS\_tapi. Пакет, который хранит процедуры create\_patient (создание пациента), update\_patient (редактирование пациента) и delete\_patient (удаление пациента).
* EMPLOYEES\_tapi. Пакет, который хранит процедуры create\_employee (создание сотрудника), update\_employee (редактирование сотрудника) и delete\_employee (удаление сотрудника).
* TALONS\_tapi. Пакет, который хранит процедуры create\_talon (создание талона), update\_talon (редактирование талона) и delete\_talon (удаление талона).
* TREATMENTS\_tapi. Пакет, который хранит процедуры create\_treatment (создание врачебного осмотра), update\_treatment (редактирование врачебного осмотра) и delete\_treatment (удаление врачебного осмотра).
* SUPPLIERS\_tapi. Пакет, который хранит процедуры create\_supplier (создание поставщика), update\_supplier (редактирование поставщика) и delete\_supplier (удаление поставщика).
* PHARMACY\_tapi. Пакет, который хранит процедуры create\_drug (создание лекарственного препарата), update\_drug (редактирование лекарственного препарата) и delete\_drug (удаление лекарственного препарата).
* POSITIONS\_tapi. Пакет, который хранит процедуры create\_position (создание должности), update\_position (редактирование должности) и delete\_position (удаление должности).
* DEPARTMENTS\_tapi. Пакет, который хранит процедуры create\_department (создание отделения), update\_department (редактирование отделения) и delete\_department (удаление отделения).
* COMMENTS\_tapi. Пакет, который хранит процедуры create\_comment (создание отзыва), update\_comment (редактирование отзыва) и delete\_comment (удаление отзыва).
* PRICELIST\_tapi. Пакет, который хранит процедуры create\_service (создание платной услуги), update\_service (редактирование платной услуги) и delete\_service (удаление платной услуги).
* DEPARTMENT\_EMPLOYEE\_tapi. Пакет, который хранит процедуры create\_department\_employee (привязать отделение к сотруднику), update\_department\_employee (изменить отделение для сотрудника) и delete\_department\_employee (удалить отношение сотрудника с отделением).
* PERSON\_ADDRESS\_tapi. Пакет, который хранит процедуры create\_person\_address (привязать персону к адресу проживания), update\_person\_address (изменить адрес проживания для персоны) и delete\_person\_address (удалить отношение персоны с адресом проживания).
* export\_josn. Экспорт данных в формат JSON.
* import\_json. Импорт данных из JSON-файла.
* book\_talon. Забронировать талон.
* unbook\_talon. Удалить бронь талона.
* compare\_passwords. Сравнение паролей.

3.1.4 Индексы

Для оптимизации времени, затрачиваемого на выполнение запросов к таблицам, были разработаны индексы.

Индексы могут быть созданы с помощь оператора CREATE INDEX. Пример создания индексов приведён в листинге 3.5.

CREATE INDEX talons\_index ON TALONS (patient\_id, employee\_id, talon\_date);

CREATE INDEX pharmacy\_index ON PHARMACY (drug, price);

CREATE INDEX treatments\_index ON TREATMENTS (patient\_id, employee\_id);

CREATE INDEX pricelist\_index ON PRICELIST (position\_id, service);

Листинг 3.5 – SQL код для создания индексов

Индексы, которые были созданы в рамках курсового проекта:

* talons\_index. Для индексирования таблицы TALONS.
* pharmacy\_index. Для индексирования таблицы PHARMACY.
* suppliers\_index. Для индексирования таблицы SUPPLIERS.
* treatments\_index. Для индексирования таблицы TREATMENTS.
* pricelist\_index. Для индексирования таблицы PRICELIST.

3.1.5 Функции

Функция в Oracle – это объект базы данных, который возвращает значение на основе переданных в нее аргументов. Функции могут быть созданы с помощью оператора CREATE FUNCTION. Пример функции, которая хеширует пароль пользователя, приведён в листинге 3.6.

CREATE OR REPLACE FUNCTION hash\_password(f\_password IN varchar2)

RETURN RAW

AS

BEGIN

IF f\_password IS NULL

THEN

RETURN NULL;

ELSE

RETURN sys.dbms\_crypto.hash(utl\_raw.cast\_to\_raw(f\_password), sys.dbms\_crypto.hash\_sh256);

END IF;

END;

Листинг 3.6 – Функция для хеширования пароля

Функции, которые были созданы в рамках курсового проекта:

* hash\_password. Хеширование пароля пользователя.

3.1.6. Триггеры

Триггер является именованным модулем PL/SQL, который хранится в базе данных и может быть вызван повторно. Вы можете включать и отключать триггер. Когда триггер включен, база данных автоматически вызывает его - то есть триггер срабатывает - всякий раз, когда происходит событие, которое вызывает триггер.

Пример реализации триггера, который отменяет бронь талона при удалении пациента, приведён в листинге 3.7.

CREATE OR REPLACE TRIGGER PATIENT\_AFTER\_DELETE

AFTER DELETE ON PATIENTS

FOR EACH ROW

BEGIN

UPDATE TALONS SET PATIENT\_ID = NULL WHERE PATIENT\_ID = :old.ID;

DELETE FROM TREATMENTS WHERE PATIENT\_ID = :old.PATIENT\_ID;

END;

Листинг 3.7 – Триггер отмены бронирования талона после удаления пациента

Триггеры, которые были созданы в рамках курсового проекта:

* PATIENT\_AFTER\_DELETE. Триггер, который подчищает данные пользователя по всей базе данных.
* EMPLOYEE\_AFTER\_DELETE. Триггер, который подчищает данные сотрудника по всей базе данных.
* USER\_AFTER\_DELETE. Триггер, который подчищает данные после пользователя по всей базе данных.

## 3.2. Описание используемой технологии

Шифрование и маскирование данных - это два разных метода защиты информации. Шифрование используется для хранения важной информации в ненадежных источниках и передачи ее по незащищенным каналам связи. Шифрование позволяет скрыть содержимое сообщения от посторонних глаз, а маскирование - скрыть часть данных от пользователей, которым эти данные не нужны. Маскирование может быть полезно, когда необходимо предоставить доступ к данным только определенным пользователям или группам пользователей.

В базе данных медицинского центра есть функция, которая хеширует пароль пользователя. Это нужно для того, чтобы не потерять критически важные данные о пользователе в случае взлома базы данных.

Хеширование пароля происходит по алгоритму SHA-256, который возвращает набор символов 16-ой системы счисления. Данный алгоритм предоставляет нам пакет DBMS\_CRYPTO. DBMS\_CRYPTO - это пакет в Oracle Database, который предоставляет интерфейс для шифрования и дешифрования хранимых данных. Он может использоваться в сочетании с программами PL/SQL, работающими с сетевыми коммуникациями. DBMS\_CRYPTO поддерживает несколько стандартных алгоритмов шифрования и хеширования, включая Advanced Encryption Standard (AES), который был одобрен Национальным институтом стандартов и технологий (NIST).

Пример использования пакета DBMS\_CRYPTO можно увидеть в листинге 3.8.

CREATE OR REPLACE FUNCTION compare\_passwords

(

user\_id in USERS.id%TYPE,

password in VARCHAR2

)

RETURN NUMBER

IS

hash RAW(32);

user\_password RAW(32);

BEGIN

SELECT password INTO user\_password FROM USERS WHERE id = user\_id;

hash := hash\_password(TRIM(password));

RETURN DBMS\_LOB.compare(hash, user\_password);

END;

Листинг 3.8 – Функция сравнения пароля с хранимым хеш-значением

Маскирование предоставляет нам пакет DBMS\_REDACT. DBMS\_REDACT - это пакет в Oracle Database, который предоставляет интерфейс для маскирования данных, возвращаемых из запросов, выполняемых низкопривилегированными пользователями или приложением. Он позволяет скрыть часть данных от пользователей, которым эти данные не нужны. DBMS\_REDACT используется для защиты конфиденциальных данных, таких как номера социального страхования, номера кредитных карт и других конфиденциальных данных. Тем самым мы улучшаем безопасность и конфиденциальность данных от пользователей, с которыми они никак не связаны.

Пример использования DBMS\_REDACT приведён в листинге 3.9.

BEGIN DBMS\_REDACT.ADD\_POLICY(

object\_schema => 'system',

object\_name => 'treatments',

column\_name => 'treatment\_info',

policy\_name => 'mask\_treatment\_info',

function\_type => DBMS\_REDACT.FULL,

expression => '1=1');

END;

Листинг 3.9 – SQL код запрета просмотра данных об осмотре пациента

Такую инструкцию довольно легко проверить. Следует подключиться к базе данных не через пользователя SYSTEM, выполнить запрос к таблице. В результате запрос вернёт строки, где определённый столбец будет скрыт.

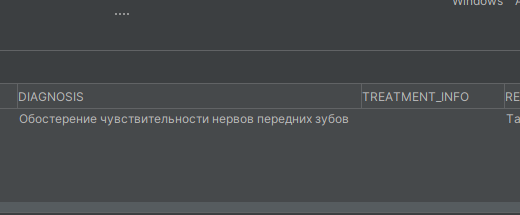


Рисунок 3.1 – Пример маскирования данных

Как можно заметить, маскирование данных на столбец TREATMENT\_INFO таблицы TREATMENTS сработало успешно.

## 3.3. Вывод

В данном разделе была разработана модель данных, была составлена UML диаграмма базы данных, а также описана каждая таблица в диаграмме. Были описаны виды разработанных объектов базы данных.

Технологии шифрования и маскирования данных достаточно просты, но одновременно так удобны и полезны в реализации по-настоящему защищённой и стойкой базы данных медицинского центра.

# 4. Установка, настройка и использование Oracle 19c

Oracle Database – это мощная реляционная СУБД, разработанная компанией Oracle Corporation. Она обеспечивает надежное хранение и управление структурированными данными, а также обладает высокой производительностью и масштабируемостью.

Oracle Database имеет широкий набор функций и возможностей, таких как многопоточная обработка, механизмы безопасности и аутентификации, индексы, триггеры, процедуры, функции и многое другое. Она используется для управления транзакционными данными, аналитическими данными, приложениями и бизнес-процессами, и является одним из основных компонентов многих крупных предприятий и организаций во всем мире.

Oracle Database поддерживает различные операционные системы и аппаратные платформы, что делает ее универсальным решением для любых задач, связанных с управлением данными.

## 4.1. Создание ролей для разграничения доступа

Роль – это набор системных и объектных прав, которые могут быть выданы и отозваны как единое целое, и после добавления этой роли, могут временно быть активированы и деактивированы во время существования сессии.

Как было указано выше, база данных медицинского центра будет иметь 4 основных роли: менеджер, врач, фармацевт и пользователь. Каждой из этой роли уже назначены те действия, которые они могут выполнять.

Первым разработаем роль пользователя. Код создания роли пользователя представлен в листинге 4.1.

CREATE ROLE "RL\_USER";

GRANT EXECUTE ON PATIENTS\_tapi TO "RL\_USER";

GRANT EXECUTE ON PERSONS\_tapi TO "RL\_USER";

GRANT EXECUTE ON PASSPORTS\_tapi TO "RL\_USER";

GRANT EXECUTE ON ADDRESSES\_tapi TO "RL\_USER";

GRANT EXECUTE ON BOOK\_TALON TO "RL\_USER";

GRANT EXECUTE ON UNBOOK\_TALON TO "RL\_USER";

GRANT EXECUTE ON PERSON\_ADDRESS\_tapi TO "RL\_USER";

GRANT EXECUTE ON COMMENTS\_tapi TO "RL\_USER";

GRANT CREATE SESSION TO "RL\_USER";

CREATE PROFILE PUSER LIMIT

FAILED\_LOGIN\_ATTEMPTS 5

PASSWORD\_LIFE\_TIME 90

PASSWORD\_GRACE\_TIME 7;

CREATE USER "USER"

IDENTIFIED BY user123

DEFAULT TABLESPACE TS\_USERS

QUOTA UNLIMITED ON TS\_USERS

PROFILE PUSER

ACCOUNT UNLOCK;

GRANT RL\_USER TO "USER";

Листинг 4.1 – Аккаунт роли пользователя

Следующей будет разработана роль доктора. Код создания роли и пользователя приведён в листинге 4.2.

CREATE ROLE RL\_DOCTOR;

GRANT EXECUTE ON TALONS\_tapi TO RL\_DOCTOR;

GRANT EXECUTE ON TREATMENTS\_tapi TO RL\_DOCTOR;

GRANT CREATE SESSION TO RL\_DOCTOR;

CREATE PROFILE PDOCTOR LIMIT

FAILED\_LOGIN\_ATTEMPTS 3

PASSWORD\_LIFE\_TIME UNLIMITED

PASSWORD\_GRACE\_TIME UNLIMITED

PASSWORD\_LOCK\_TIME UNLIMITED;

CREATE USER "DOCTOR"

IDENTIFIED BY doctor123

DEFAULT TABLESPACE TS\_USERS

QUOTA UNLIMITED ON TS\_USERS

PROFILE PDOCTOR

ACCOUNT UNLOCK;

GRANT RL\_DOCTOR TO "DOCTOR";

Листинг 4.2 – Аккаунт роли врача

Следующей будет разработана роль фармацевта. Код создания роли и пользователя приведён в листинге 4.3.

CREATE ROLE RL\_PHARMACIST;

GRANT EXECUTE ON SUPPLIERS\_tapi TO RL\_PHARMACIST;

GRANT EXECUTE ON PHARMACY\_tapi TO RL\_PHARMACIST;

GRANT CREATE SESSION TO RL\_PHARMACIST;

CREATE PROFILE PPHARMACIST LIMIT

FAILED\_LOGIN\_ATTEMPTS 3

PASSWORD\_LIFE\_TIME UNLIMITED

PASSWORD\_GRACE\_TIME UNLIMITED

PASSWORD\_LOCK\_TIME UNLIMITED;

CREATE USER "PHARMACIST"

IDENTIFIED BY phar123

DEFAULT TABLESPACE TS\_USERS

QUOTA UNLIMITED ON TS\_USERS

PROFILE PPHARMACIST

ACCOUNT UNLOCK;

GRANT RL\_PHARMACIST TO "PHARMACIST";

Листинг 4.3 – Аккаунт роли фармацевта

Последней будет реализована роль менеджера. Код создания роли и пользователя приведён в листинге 4.4.

CREATE ROLE RL\_MANAGER;

GRANT RL\_USER TO RL\_MANAGER;

GRANT RL\_PHARMACIST TO RL\_MANAGER;

GRANT RL\_DOCTOR TO RL\_MANAGER;

GRANT EXECUTE ON EMPLOYEES\_tapi TO RL\_MANAGER;

GRANT EXECUTE ON POSITIONS\_tapi TO RL\_MANAGER;

GRANT EXECUTE ON DEPARTMENTS\_tapi TO RL\_MANAGER;

GRANT EXECUTE ON DEPARTMENT\_EMPLOYEE\_tapi TO RL\_MANAGER;

GRANT EXECUTE ON PRICELIST\_tapi TO RL\_MANAGER;

GRANT CREATE SESSION,

CREATE PROCEDURE,

CREATE TABLE,

CREATE TRIGGER TO RL\_MANAGER;

CREATE PROFILE PMANAGER LIMIT

FAILED\_LOGIN\_ATTEMPTS 3

PASSWORD\_LIFE\_TIME UNLIMITED

PASSWORD\_GRACE\_TIME UNLIMITED

PASSWORD\_LOCK\_TIME UNLIMITED;

CREATE USER "MANAGER"

IDENTIFIED BY manager123

DEFAULT TABLESPACE TS\_USERS

QUOTA UNLIMITED ON TS\_USERS

PROFILE PMANAGER

ACCOUNT UNLOCK;

GRANT RL\_MANAGER TO "MANAGER";

Листинг 4.4 – Аккаунт роли менеджера

Все пользователи были успешно созданы и проверены на их корректную работоспособность с пакетами и процедурами, привязанных к ним.

## 4.2. Описание процедур экспорта и импорта данных

Процедуры импорта и экспорта являются инструментами для деления данными между различными базами данных или для создания резервных копий базы данных.

JSON (JavaScript Object Notation) – это легкий формат обмена данными, который используется для представления структурированных данных в виде пар "имя/значение". Он широко используется в приложениях и в базах данных.

Использование формата JSON для импорта и экспорта данных предоставляет удобный и гибкий способ обмена данными между различными базами данных или веб-приложениями. Он позволяет представлять структурированные данные в виде пар "имя/значение" и легко преобразовывать данные в различные форматы.

В листинге 4.5 приведена процедура, который получает данные из таблицы, объединяет их в один массив и сохраняет в файл.

CREATE OR REPLACE PROCEDURE EXPORT\_JSON

IS

v\_file UTL\_FILE.FILE\_TYPE;

v\_cursor SYS\_REFCURSOR;

v\_row COMMENTS%ROWTYPE;

v\_json CLOB;

BEGIN

v\_file := UTL\_FILE.FOPEN('UTL\_DIR', 'comments.json', 'W');

OPEN v\_cursor FOR SELECT \* FROM SYSTEM.COMMENTS;

v\_json := '[';

LOOP

FETCH v\_cursor INTO v\_row;

EXIT WHEN v\_cursor%NOTFOUND;

IF v\_json != '[' THEN

v\_json := v\_json || ',';

END IF;

v\_json := v\_json || '{';

v\_json := v\_json || '"user\_id":' || v\_row.user\_id || ',';

v\_json := v\_json || '"comment\_text":"' || REPLACE(v\_row.comment\_text, '"', '\"') || '",';

v\_json := v\_json || '"employee\_id":"' || REPLACE(v\_row.employee\_id, '"', '\"') || '"';

v\_json := v\_json || '}';

END LOOP;

CLOSE v\_cursor;

v\_json := v\_json || ']';

UTL\_FILE.PUT\_LINE(v\_file, v\_json);

UTL\_FILE.FCLOSE(v\_file);

END;

Листинг 4.5 – Процедура экспорта данных в json-файл

В листинге 4.6 приведена процедура, которая считывает данные из json-файла и заносит их в нужную таблицу:

CREATE OR REPLACE PROCEDURE IMPORT\_JSON

IS

BEGIN

INSERT INTO COMMENTS (user\_id, comment\_text, employee\_id)

SELECT user\_id, comment\_text, employee\_id

FROM JSON\_TABLE(BFILENAME('UTL\_DIR', 'COMMENTS.JSON'), '$[\*]' COLUMNS (

user\_id INTEGER PATH '$.user\_id',

comment\_text VARCHAR2(1024) PATH '$.comment\_text',

employee\_id INTEGER PATH '$.employee\_id'

)

);

END;

Листинг 4.6 – Процедура импорта данных из json-файла

Процедуры экспорта и импорта успешно скомпилировались базой данных, что говорит о том, что они написаны по крайней мере без синтаксических ошибок.

## 4.3. Заполнение таблицы 100000 строк

В листинге 4.7 представлена процедура, которая генерирует 100000 строк для таблицы SUPPLIERS.

--/

CREATE OR REPLACE PROCEDURE generate\_suppliers

IS

BEGIN

for i in 1..100000 loop

insert into suppliers (supplier\_name, supplier\_country)

values (

case floor(dbms\_random.value(1, 31)) -- Generate a random name

when 1 then 'Roche'

when 2 then 'Novartis'

when 3 then 'Merck'

when 4 then 'AbbVie'

when 5 then 'Janssen'

when 6 then 'GlaxoSmithKline'

when 7 then 'Bristol Myers Squibb'

when 8 then 'Pfizer'

when 9 then 'Sanofi'

when 10 then 'Takeda'

when 11 then 'AstraZeneca'

when 12 then 'Gilead'

when 13 then 'Lilly'

when 14 then 'Amgen'

when 15 then 'Bayer'

when 16 then 'Novo Nordisk'

when 17 then 'Boehringer Ingelheim'

when 18 then 'Teva'

when 19 then 'Biogen'

when 20 then 'Viatris'

when 21 then 'Roche Pharma'

when 22 then 'Novartis Oncology'

when 23 then 'Merck Serono'

when 24 then 'Abbott'

when 25 then 'Johnson & Johnson'

when 26 then 'GlaxoSmithKline Consumer Healthcare'

when 27 then 'Bristol Myers Squibb India'

when 28 then 'Pfizer Consumer Healthcare'

when 29 then 'Sanofi Pasteur'

when 30 then 'Takeda Oncology'

end,

case floor(dbms\_random.value(1, 21)) -- Generate a random country

when 1 then 'Belarus'

when 2 then 'Russia'

when 3 then 'Ukraine'

when 4 then 'Poland'

when 5 then 'Lithuania'

when 6 then 'Germany'

when 7 then 'France'

when 8 then 'China'

when 9 then 'Italy'

when 10 then 'Spain'

when 11 then 'United Kingdom'

when 12 then 'United States'

when 13 then 'Canada'

when 14 then 'Brazil'

when 15 then 'India'

when 16 then 'Japan'

when 17 then 'Australia'

when 18 then 'South Africa'

when 19 then 'Sweden'

when 20 then 'Norway'

end

);

end loop;

commit;

END;

--/

Листинг 4.7 – Процедура заполнения данными таблицы SUPPLIERS

Процедура добавления случайно сгенерированных данных в таблицу SUPPLIERS успешно скомпилировалась, что говорит на о том, что они написаны по крайней мере без синтаксических ошибок

## 4.4. Тестирование производительности базы данных

Тестирование производительности базы данных – это процесс измерения и анализа производительности базы данных при выполнении различных операций и запросов. Это позволяет убедиться, что база данных способна обрабатывать запросы пользователей с достаточной скоростью и производительностью, чтобы поддерживать высокую нагрузку.

Первым делом протестируем таблицу SUPPLIERS. После того, как была выполнена процедура на заполнение данными, в таблице есть 100000 строк. Построим простой запрос на получение всех поставщиков, которые находятся в определённой стране и просмотрим время выполнения.



Рисунок 4.1 – Время выполнения запроса без индекса

Теперь добавим индекс на эту таблицу по столбцу страны поставщика и посмотрим на результат выполнения запроса.



Рисунок 4.2 – Время выполнения запроса с индексом

Как можно заметить, время выполнения запроса уменьшилось, однако не сильно много. Попробуем изменить запрос на несколько условий, где будет проверка не только на название страны, но ещё и на имя поставщика. Имеем следующий код, который предоставлен в листинге 4.8

SELECT \* FROM SYSTEM.SUPPLIERS

WHERE (SUPPLIERS.SUPPLIER\_NAME LIKE '%i%' OR SUPPLIERS.SUPPLIER\_NAME LIKE '%a%') and

(SUPPLIERS.SUPPLIER\_COUNTRY LIKE '%u%' or SUPPLIERS.SUPPLIER\_COUNTRY LIKE '%e%');

Листинг 4.8 – Запрос к таблице SUPPLIERS с несколькими условиями

Как и ожидалось, время запроса с несколькими условиями будет не сильно, но всё таки больше.



Рисунок 4.3 – Время выполнения нового запроса без индексов

Теперь добавим индекс только на один столбец, а затем на два столбца и посмотрим разницу во времени.



Рисунок 4.4 – Время выполнения нового запроса с индексом на один столбец

Теперь оценим время выполнения запроса с наличием комбинированного индекса.

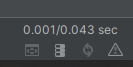


Рисунок 4.5 – Время выполнения нового запроса с комбинированным индексом

Как мы можем заметить, в некоторых моментах индекс реально улучшает скорость выполнения запроса, что не может не радовать. Данные показатели хоть и были проведены на достаточно маленько й таблице, однако можно смело утверждать, что индексы реально помогут улучшить производительность базы данных медицинского центра в будущем.

## 4.5. Вывод

Всю эту главу можно описать как некая работа над безопасностью и производительностью БД. В ходе анализа того, что хранит база данных медицинского центра, был выявлены роли, которые как нельзя лучше подходят по специфике задач и возможностей. Чтобы каждый делал только, что ему дозволено, были созданы роли – поименованные наборы разрешений. Для каждой роли были созданы свои собственные объекты пользователей БД, которые смогу подключаться к ней и выполнять только требуемые от них задачи.

Чтобы каждый такой пользователь мог выполнять свои задачи и не волноваться о проблемах эффективности обработки данных, было принято решение сделать индексирование таблиц, что реально улучшает скорость чтения данных. Доказано путём проведения анализа времени работы ёмкого запроса к таблице, которая состоит из 100000 строк. Опыты показали положительные результаты.

# 5. Тестирование, проверка работоспособности и анализ полученных данных

В ходе этой главы будут протестированы возможности все те возможности, которые были описаны в предыдущих главах. В приоритете ­– проверка на шифрование и маскирование данных.

## 5.1. Тестирование клиентской части

Пользователь имеет права на создание и редактирование аккаунта, создание и редактирование пациента, на выполнение процедур бронирования и отмены бронирования талона.

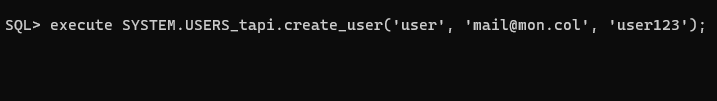


Рисунок 5.1 – Создание аккаунта у пользователя

Теперь произведём проверку того, хранится ли результат процедуры в базе данных.



Рисунок 5.2 – Новая запись в базе данных

Здесь сразу можно наблюдать, что шифрование (хеширование) работает отлично, и никто не сможет узнать пароль пользователей в случае взлома.

Далее проверим работу бронирования талона.

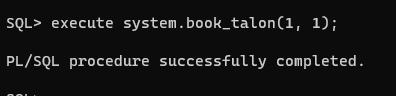


Рисунок 5.3 – Процедура бронирования талона

Теперь произведём проверку того, хранится ли результат процедуры в базе данных.

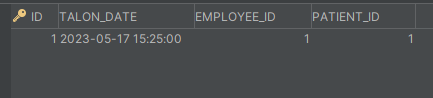


Рисунок 5.4 – Запись талона в базе данных обновилась

Как можно заметить, функционал пользователя работает отлично без проблем и ошибок. Тестирование показало отличные результаты.

## 5.2. Тестирование фармацевтической части

Роль фармацевта управляет данными аптеки медицинского центра, а значит именно фармацевт добавляет данные в таблицы поставщиков лекарств и медицинских препаратов.

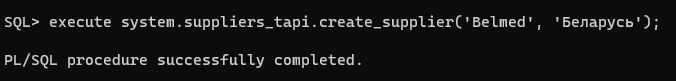


Рисунок 5.5 – Выполнение процедуры на создание поставщика

Теперь произведём проверку того, хранится ли результат процедуры в базе данных.

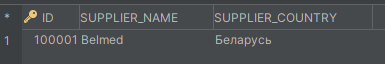


Рисунок 5.6 – Новая запись в таблице поставщиков

Теперь создадим лекарственный препарат и привяжем к нему ранее созданного поставщика.

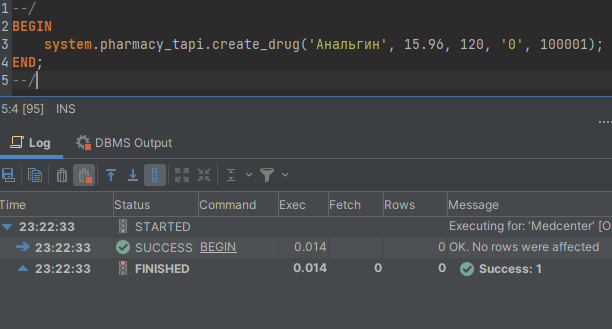


Рисунок 5.7 – Успешное выполнение процедуры

Теперь произведём проверку того, хранится ли результат процедуры в базе данных.

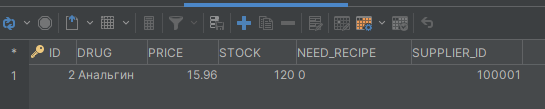


Рисунок 5.8 – Результат процедуры

Тестирование функционала фармацевта медицинского центра проведено успешно, без различных ошибок.

## 5.3. Тестирование врачебной части

Первым делом рассмотрим процедуру на создании талона к врачу.

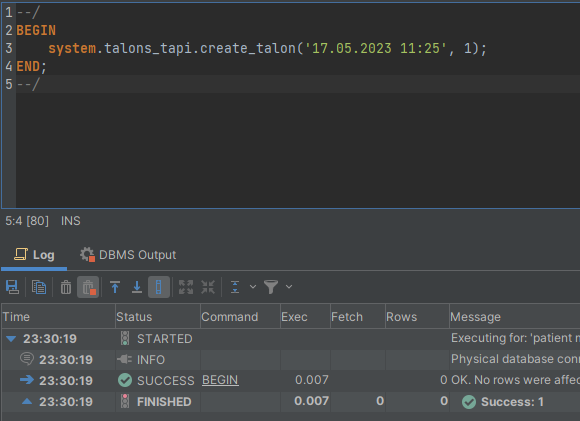


Рисунок 5.9 – Успешно выполнение процедуры

Теперь проверим наличие созданного процедурой талона в таблице талонов в базе данных

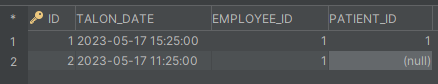


Рисунок 5.10 – Наличие талона в базе данных

Далее проверим удаление талона, ибо врач может не только их создавать, но также и изменять, и удалять.

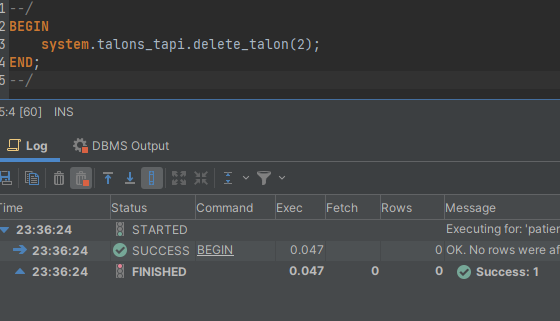


Рисунок 5.11 – Успешное выполнение процедуры

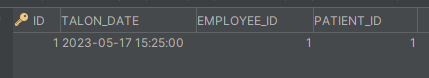


Рисунок 5.12 – Изменение таблицы после удаления талона

Тестирование функционала врача медицинского центра проведено успешно, без различных ошибок.

## 5.4. Тестирование части менеджера

Последним на очереди остаётся менеджер. Этот пользователь, как было описано выше, имеет доступ ко всем возможным данным и процедурам.

Первым делом протестируем создание должности сотрудников медицинского центра.

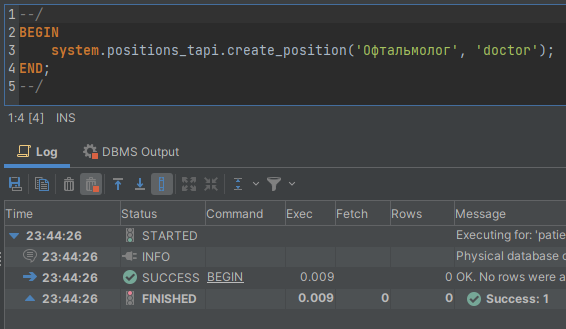


Рисунок 5.13 – Успешное выполнение процедуры

Теперь проверим наличие новой должности, которую создал менеджер базы данных.

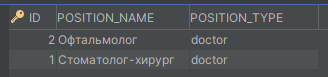


Рисунок 5.14 – Наличие записи новой должности

Так, как менеджер может делать то, что и другие пользователи, тогда проверим выполнение процедур на изменение талонов врачей.

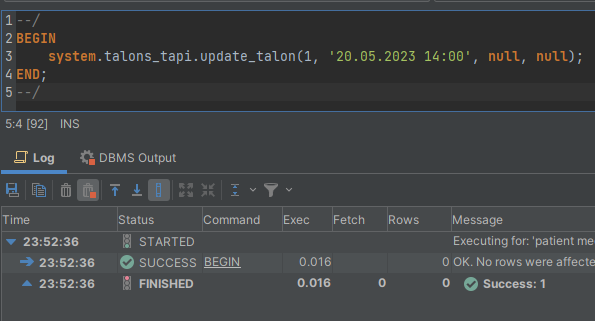


Рисунок 5.15 – Успешное выполнение процедуры

Теперь проверим наличие изменений талона с идентификатором 1. Стоит упомянуть, что если в некую процедуру обновления вставить параметр null, то значение столбца буде оставаться неизменным. В случае, как выше, если бы были установлены некоторое значение

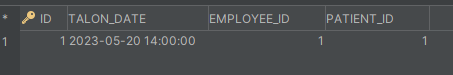


Рисунок 5.16 – Наличие измененной даты приема талона

Тестирование функционала менеджера медицинского центра проведено успешно, без различных ошибок.

## 5.5. Проверка работоспособности процедур импорта и экспорта

Сперва выполним проверку работы экспорта данных в json-файл.

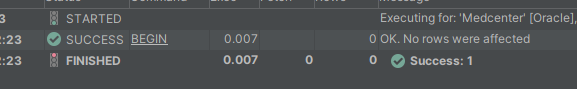


Рисунок 5.17 – Успешное выполнение процедуры экспорта

Теперь проверим, есть ли данные в самом json-файле, куда процедура должна была сохранить данные.

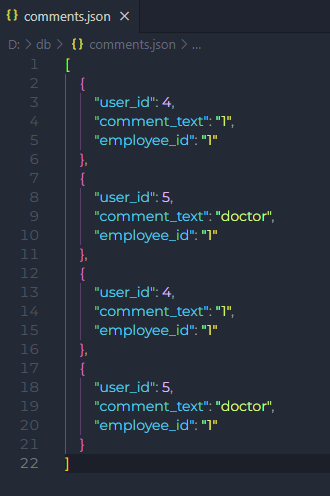


Рисунок 5.18 – Содержание json-файла после экспорта

Как можно заметить, процедура экспорта работает отлично. Теперь проверим выполнение процедуры импорта данных, но перед этим заменим некоторое значение в json-файле, чтобы точно наблюдались изменения базы данных.

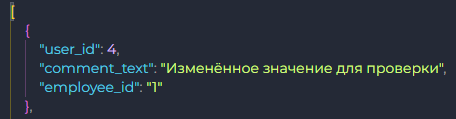


Рисунок 5.19 – Содержимое json-файла

Теперь выполним процедуру и проверим правильно ли вставились данные в таблицу. По подсчётам, строк должно выйти ровно 8.

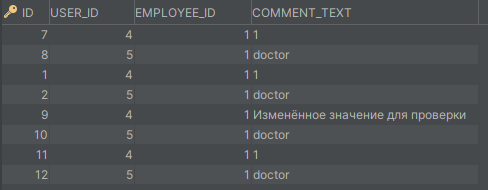


Рисунок 5.20 – Значения данных после импорта

Как можно заметить, строк ровно 8. А также изменённая строка также здесь. Можно смело утверждать, что процедуры для экспорта и импорта работают отлично и без всяких проблем.

## 5.6. Вывод

В главе были показаны возможности каждого пользователя базы данных. Как можно было наблюдать, все процедуры выполнялись так, как и ожидалось от них. Те процедуры, которые не были указаны в подразделах были успешно проверены. Каждая из этих процедур прошла проверку на работоспособность. Если какая-то процедура всё же не прошла проверку, то она оперативно исправлялась.

Если говорить в общем, то можно сделать следующий вывод: наличие отлично построенных объектов базы данных (процедур, функций, индексов и т.д.) показывает отличный уровень эффективности в работе с самой базой данных.

# 6. Руководство по использованию программного средства

## 6.1. Установка приложения

Установка приложения довольно комплексная, хотя и не очень тяжелая. Первым делом требуется скачать PHP, так как фреймворк написан на этом языке.

Далее после установки PHP, требуется настроить в файле конфигураций подключение библиотек, которые будут отвечать за соединение и работу с базой данных Oracle 19c.

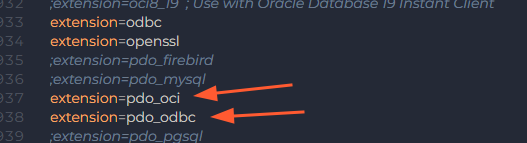


Рисунок 6.1 – Файл конфигурации PHP

Так, как Yii2 поставляется через репозитории, то для этого нужно установить пакетный менеджер Composer. После его установки достаточно выполнить простую команду, которая загрузит образ фреймворка на компьютер. Команда загрузки Yii2 приведён в листинге 6.1.

composer create-project --prefer-dist yiisoft/yii2-app-basic basic

Листинг 6.1 – Команда загрузки Yii2 через composer

Следующим шагом является установка nginx. Nginx на windows поставляется архивом, поэтому его просто достаточно разархивировать в требуемую папку. После требуется настроить конфигурацию веб-сервера для работы с приложением. Некоторые моменты настройки конфигураций представлены на изображении ниже.

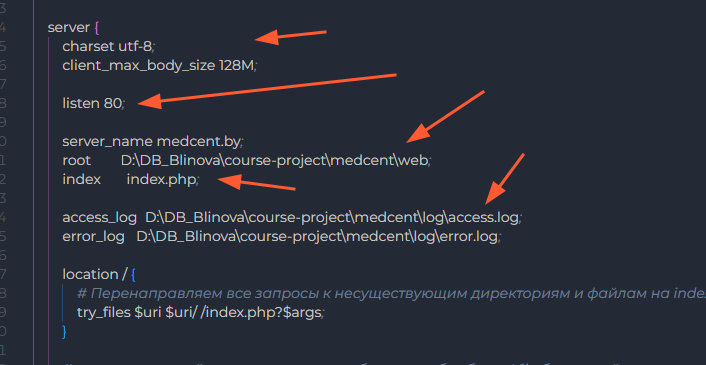


Рисунок 6.2 – Содержимое файла конфигурации nginx

После правильной настройки всех параметров PHP и nginx, заходим на адрес локального хоста с портом 80. Если на странице есть слово «Congratualtions», значит фреймворк установлен правильно.

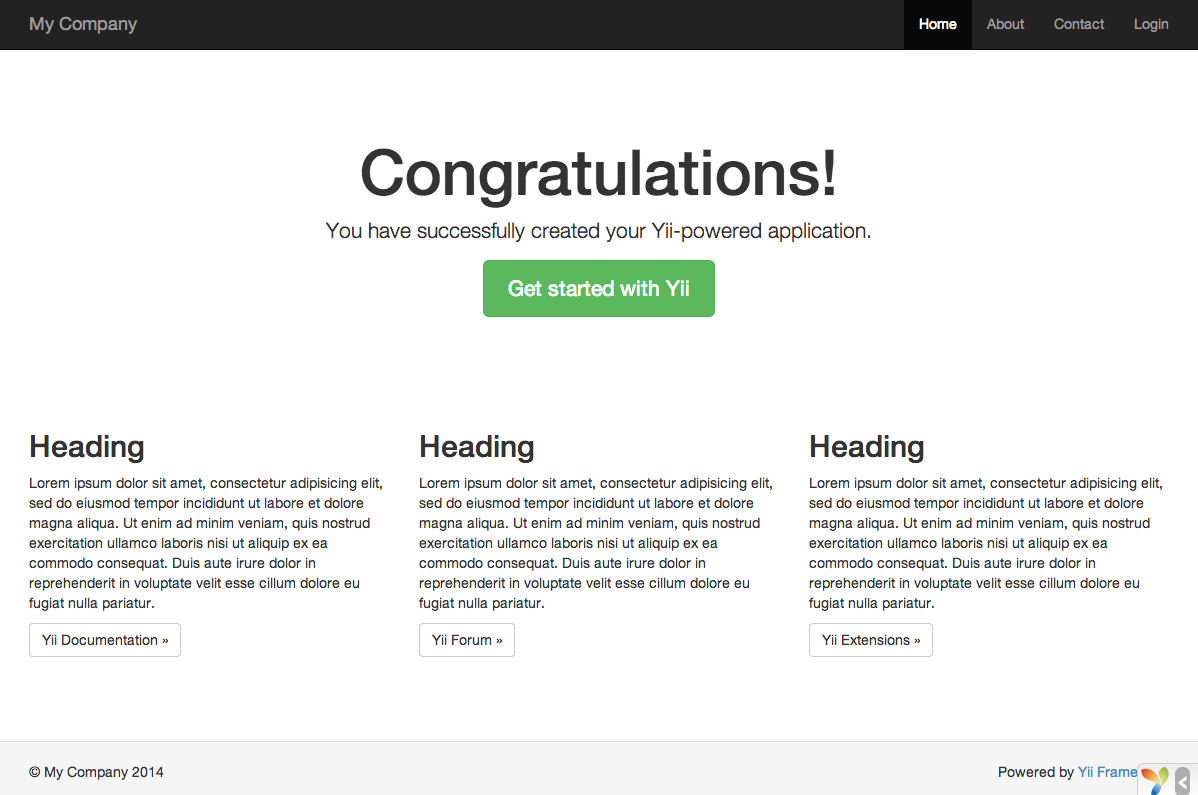


Рисунок 6.3 – Главная страница фреймворка Yii2

Описание установки довольно простое и понятное. Чтобы установить приложение, которое было создано в рамках курсового проекта, требуется скачать архив с этим приложением с GitHub репозитория [16], а дальше сделать так, как показано выше.

## 6.2. Панель менеджера

Менеджер имеет свою панель администрирования, которая выполнена довольно удобно.

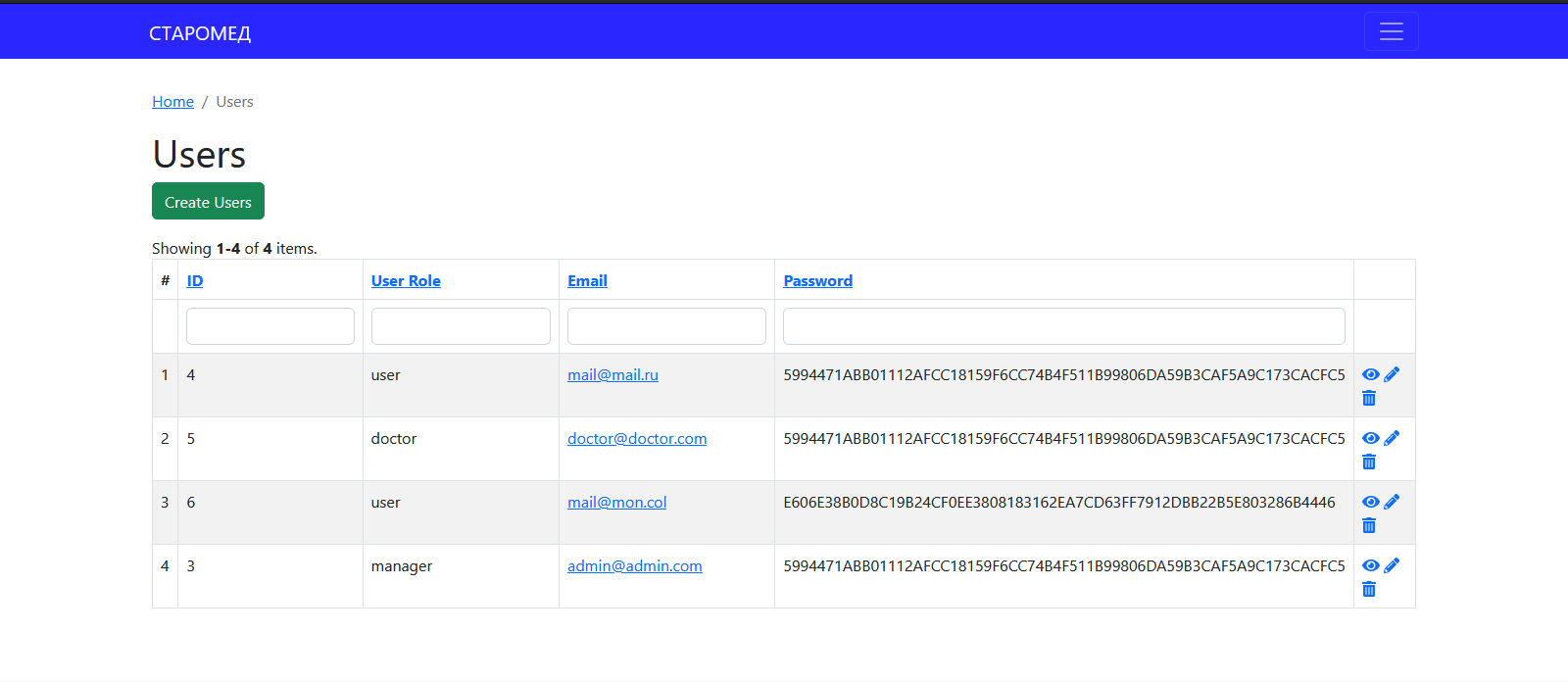


Рисунок 6.4 – Панель менеджера

Так, как база данных имеет не одну таблицу, а 17, то и страниц для управления данными сущностей тоже должно быть 17.



Рисунок 6.5 – Меню сущностей менеджера

Все возможные операции над объектами базы данных были выполнены с помощью встроенного генератора Gii.

## 6.3. Клиентская часть приложения

На рисунке ниже представлена страница регистрации пользователя.

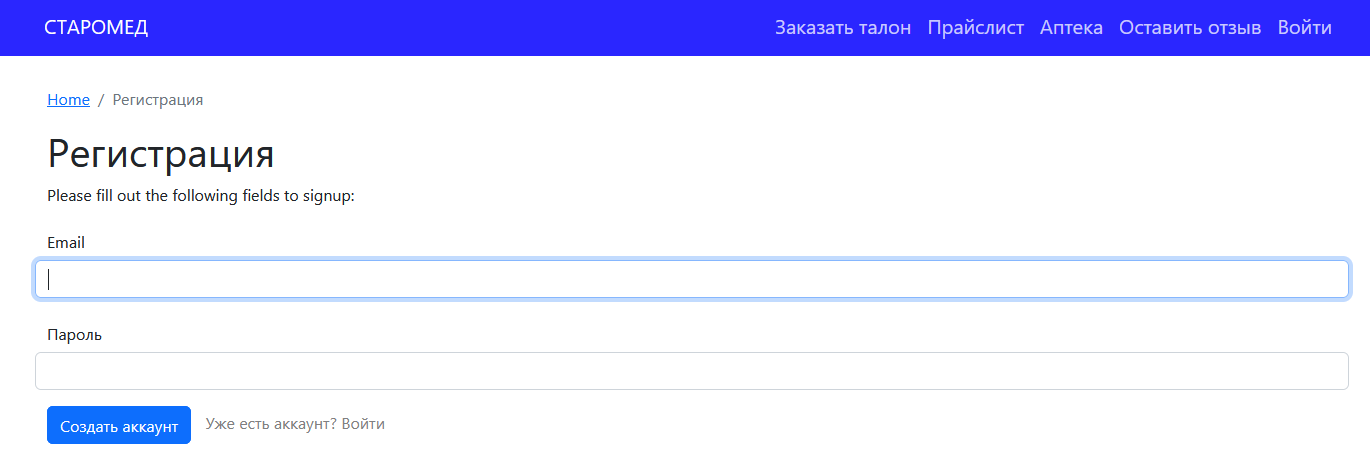


Рисунок 6.6 – Страница регистрации

На следующем изображении представлена страница авторизации пользователей.

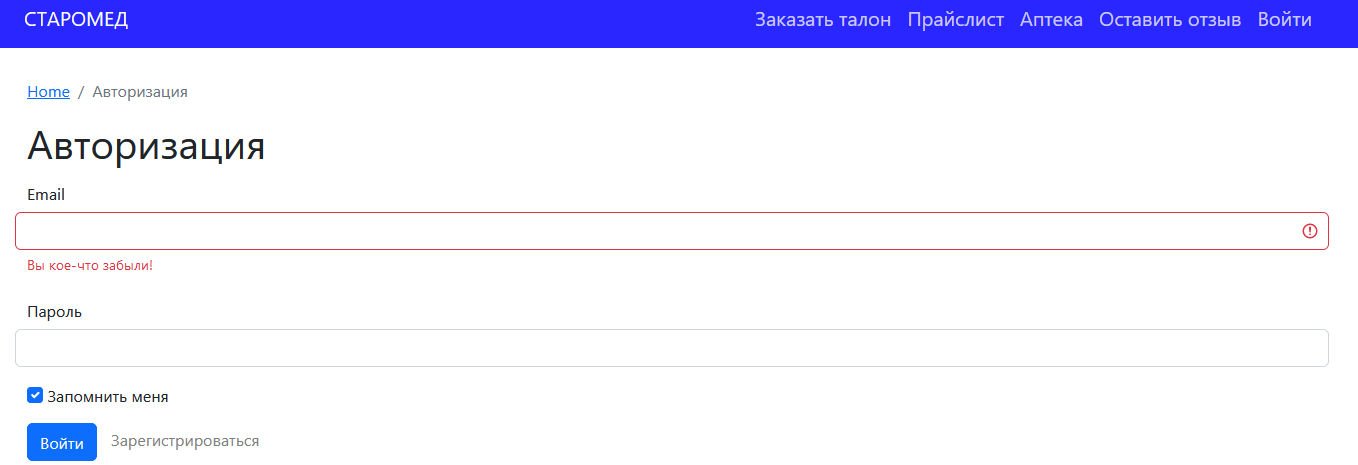


Рисунок 6.7 – Страница авторизации

На следующем изображении находиться главная страница аккаунта пользователя, на которой расположены забронированные талоны к врачам.



Рисунок 6.8 – Главная страница аккаунта пользователя

Так, как реализована возможность на один аккаунт создавать несколько пациентов, то для удобства пользователя следует вывести список его собственных пациентов. Страница пациентов пользователя изображена на рисунке ниже.

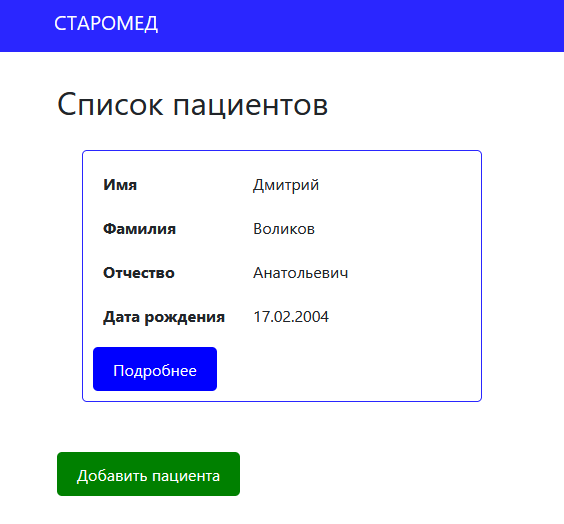


Рисунок 6.9 – Страница пациентов пользователя

Пользователь может изменять данные каждому своему пациенту, для этого предусмотрены несколько страниц с формами.



Рисунок 6.10 – Страница изменения персональных данных пациента

Пациенту можно привязать адрес, так как многие медицинские структуры требуют хранения такого рода информации.

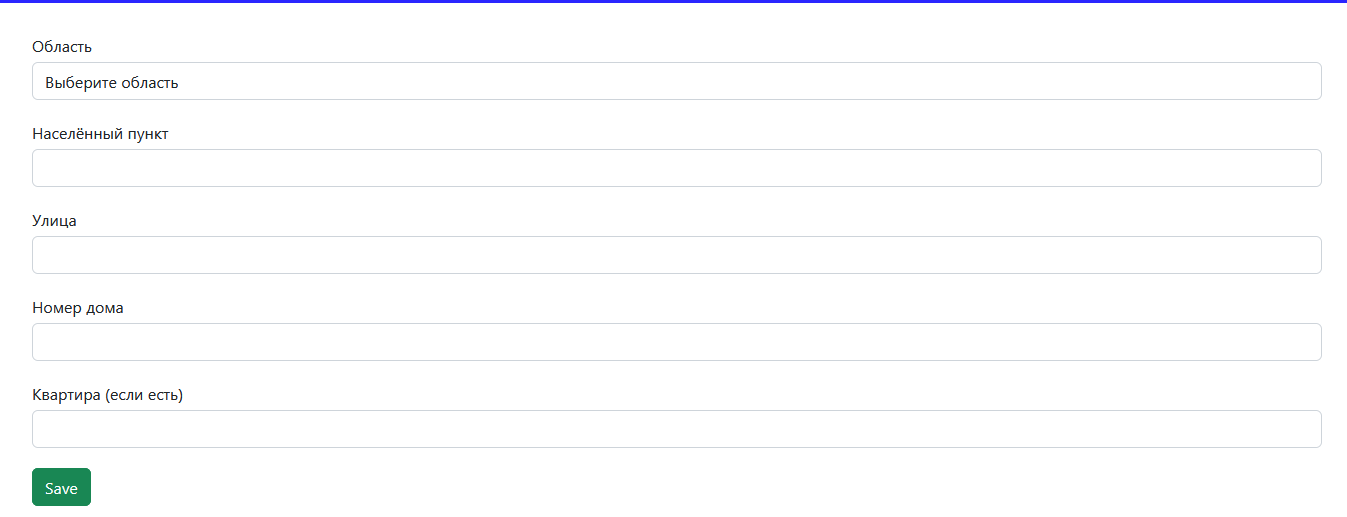


Рисунок 6.11 – Форма добавления адреса к пациенту

Ну и обязательно должна быть страничка, которая выводит информацию о свободных талонах к врачу.

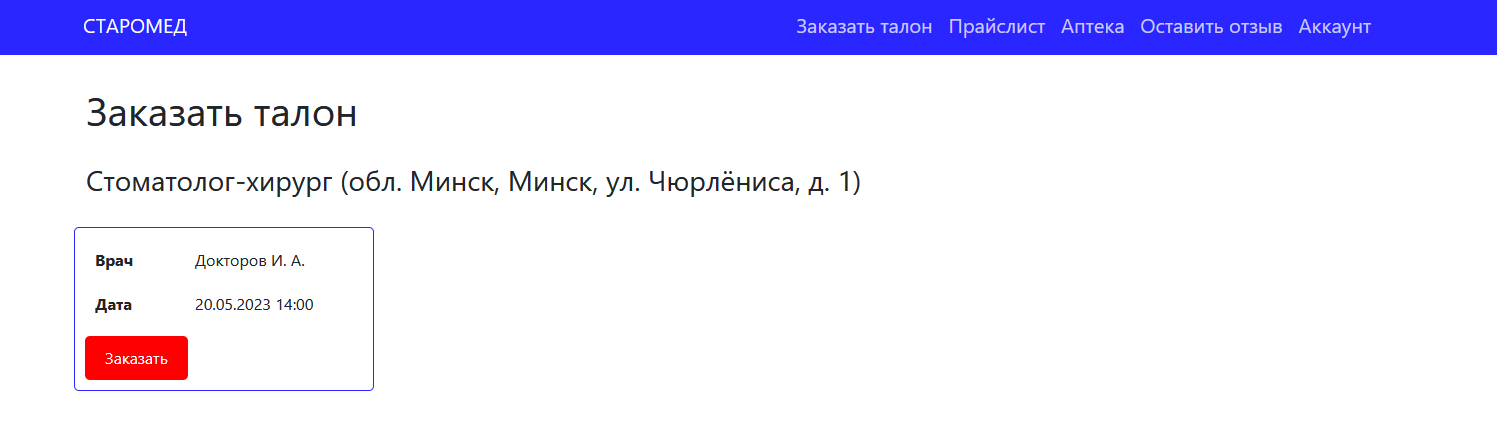


Рисунок 6.12 – Страница доступных талонов

Как можно заметить, приложение имеет самый простой функционал, который был разработан для удобства пользователя при работе с самим приложением.

## 6.3. Вывод

Приложение было создано довольно быстро, но было учтено довольно много моментов. Пожалуй, самый основной момент – это разделение обязанностей каждому пользователю. Панель менеджера доступна только самому менеджеру. Пользователю же доступны операции бронирования талонов и добавления пациентов.

В заключении стоит добавить, что разработка подобного рода приложений – это очень тяжелая работа. Поэтому были придуманы фреймворки. Они очень сильно упрощают время подъёма приложения, что позволяет любому разработчику в короткие сроки закончить проект.

Заключение

Проектирование и разработка базы данных – работа не самая лёгкая. Однако такого рода занятия задачи очень сильно качают навыки как разработчика, так и проектировщика.

В данной работе была реализована база данных медицинского центра с использованием СУБД Oracle и средств шифрования и маскирования. Целью работы было создание эффективной и надежной базы данных, которая позволит эффективно управлять данными о пациентах, сотрудниках, талонов, отделений, аптеки и должностях.

На основе разработанной модели были созданы необходимые объекты базы данных, такие как таблицы, триггеры, индексы, табличные пространства, пакеты, процедуры, функции и ограничения. Использовались инструменты DBMS\_CRYPTO и DBMS\_REDACT для реализации шифрования и маскирования данных.

Важным этапом работы было тестирование производительности базы данных. Были проведены тесты выполнения запросов. Результаты тестирования показали хорошие оценки времени обработки информации в том или ином образе.

База дынных медицинского центра имеет разделение обязанностей по ролям. Было выделено 4 роли: пользователь, врач, фармацевт и менеджер. Каждой из этих ролей присваивались свои собственные разрешения на выполнение некоторых операции над объектами базы данных.

Как итог, работа позволила создать удобную в использовании и гибкую в управлении базу данных, которая может быть использована для управления данными местной поликлиники, а также может быть адаптирована под конкретные потребности пользователей.

# Список использованных литературных источников

1. Talon.by | Заказ талонов к врачу через интернет [Электронный адрес] / Режим доступа: https://talon.by/ – Дата доступа: 18.03.2023
2. Talon.by – запись к врачу [Электронный ресурс] / Режим доступа: https://play.google.com/store/apps/details?id=by.talon.app.twa&hl=ru&gl=US – Дата доступа 09.04.2023.
3. Городская студенческая поликлиника [Электронный ресурс] / Режим доступа: https://33gsp.by/ – Дата доступа 18.03.2023
4. 3-я центральная районная клиническая поликлиника [Электронный ресурс] / Режим доступа: https://www.3crkp.by/ – Дата доступа 18.03.2023
5. Форум StackOverflow [Электронный ресурс] / Справочник Режим доступа: https://ru.stackoverflow.com/ – Дата доступа 02.04.2023.
6. Оффициальный сайт Oracle [Электронный ресурс] / Справочник Режим доступа: https://www.oracle.com/ – Дата доступа 6.04.2023.
7. Оффициальная документация Oracle [Электронный ресурс] / Справочник Режим доступа: https://docs.oracle.com/en/ – Дата доступа 12.04.2023.
8. Информационный портал Oracle-patches [Электронный ресурс] / Справочник Режим доступа: https://oracle-patches.com/ – Дата доступа 18.04.2023.
9. Информационный портал Oracle-dba [Электронный ресурс] / Справочник Режим доступа: https://oracle-dba.ru/docs – Дата доступа 24.04.2023.
10. Документация по Oracle PL/SQL [Электронный ресурс] / Справочник Режим доступа: https://oracleplsql.ru/ – Дата доступа 26.04.2023.
11. Yii2 framework [Электронный ресурс] / Режим доступа: https://www.yiiframework.com/ – Дата доступа 25.04.2023.
12. Официальная документация Yii2 [Электронный ресурс] / Режим доступа: https://www.yiiframework.com/doc/guide/2.0/ru – Дата доступа: 25.04.2023.
13. Официальный форум разработчиков Yii2 [Электронный ресурс] / Режим доступа: https://forum.yiiframework.com/ – Дата доступа: 25.04.2023.
14. Хабр Q&A [Элеткронный ресурс] / Режим доступа: https://qna.habr.com/ – Дата доступа: 10.04.2023.
15. Oracle Tutorial [Электронный ресурс] / Режим доступа: https://www.oracletutorial.com/ – Дата доступа: 14.05.2023
16. 0ero-1ne Oracle Medcenter App [Электронный ресурс] / Режим доступа: https://github.com/0ero-1ne/oracle-medcenter – Дата доступа: 10.05.2023.

# Приложение А

Структура базы данных

CREATE TABLE ROLES

(

role\_name NVARCHAR2(64) PRIMARY KEY

) TABLESPACE TS\_USERS;

Листинг А.1 – Таблица ROLES

CREATE TABLE ADDRESSES

(

id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

region NVARCHAR2(64) NOT NULL,

town NVARCHAR2(64) NOT NULL,

street NVARCHAR2(64) NOT NULL,

house\_number NVARCHAR2(10) NOT NULL, *-- VARCHAR2 BECAUSE MAY BE NOT ONLY NUMBERS, BUT ANOTHER AS "-, a-z, A-Z" (80B)*

flat NVARCHAR2(10) *-- VARCHAR2 BECAUSE MAY BE NOT ONLY NUMBERS, BUT ANOTHER AS "-, a-z, A-Z" (301-A)*

) TABLESPACE TS\_USERS;

Листинг А.2 – Таблица ADDRESSES

CREATE TABLE PASSPORTS

(

id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

passport\_number NVARCHAR2(64) NOT NULL UNIQUE, *-- IDENTIFICATION NUMBER*

date\_of\_issue DATE NOT NULL,

date\_of\_expiry DATE NOT NULL,

authority NVARCHAR2(256) NOT NULL *-- THE AUTHORITY THAT ISSUED THE PASSPORT*

) TABLESPACE TS\_USERS;

Листинг А.3 – Таблица PASSPORTS

CREATE TABLE USERS(

id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

user\_role NVARCHAR2(64) NOT NULL,

email NVARCHAR2(256) NOT NULL UNIQUE,

password RAW(32) NOT NULL,

CONSTRAINT FK\_USER\_ROLE FOREIGN KEY (user\_role)REFERENCES ROLES (role\_name)) TABLESPACE TS\_USERS;

Листинг А.4 – Таблица USERS

CREATE TABLE PERSONS

(

id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

first\_name NVARCHAR2(64) NOT NULL, *-- COLUMN FOR NAME*

second\_name NVARCHAR2(64) NOT NULL, *-- COLUMN FOR SURNAME*

last\_name NVARCHAR2(64), *-- COLUMN FOR PATRONOMYC (USUALLY FOR CIS)*

passport\_id INTEGER UNIQUE,

birth\_date DATE NOT NULL,

gender CHAR(1) NOT NULL,

CONSTRAINT FK\_PATIENT\_PASSPORT

FOREIGN KEY (passport\_id)

REFERENCES PASSPORTS(id),

CONSTRAINT CHK\_PATIENT\_GENDER CHECK (gender IN ('m','f'))

) TABLESPACE TS\_USERS;

Листинг А.5 – Таблица PERSONS

CREATE TABLE PERSON\_ADDRESS

(

id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

person\_id INTEGER NOT NULL,

address\_id INTEGER NOT NULL,

CONSTRAINT FK\_PERSON\_ADDRESS\_PERSON

FOREIGN KEY (person\_id)

REFERENCES PERSONS(id),

CONSTRAINT FK\_PERSON\_ADDRESS\_ADDRESS

FOREIGN KEY (address\_id)

REFERENCES ADDRESSES(id)

) TABLESPACE TS\_USERS;

Листинг А.6 – Таблица PERSON\_ADDRESS

CREATE TABLE PATIENTS

(

id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

auth\_data INTEGER NOT NULL,

person\_id INTEGER NOT NULL,

phone NVARCHAR2(20) NOT NULL,

CONSTRAINT FK\_PATIENT\_AUTHDATA

FOREIGN KEY (auth\_data)

REFERENCES USERS(id),

CONSTRAINT FK\_PATIENT\_PERSON

FOREIGN KEY (person\_id)

REFERENCES PERSONS(id)

) TABLESPACE TS\_MEDCENTER;

Листинг А.7 – Таблица PATIENTS

CREATE TABLE POSITIONS

(

id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

position\_name NVARCHAR2(256),

position\_type NVARCHAR2(20), *-- IN THE MEDICAL DEPARTMENT THERE ARE NOT ONLY DOCTORS, BUT ALSO CLEANERS, PROGRAMMERS, ETC.*

CONSTRAINT CHK\_POSITION\_TYPE CHECK (position\_type IN ('doctor', 'head\_doctor', 'programmer', 'cleaner', 'security'))

) TABLESPACE TS\_MEDCENTER;

Листинг А.8 – Таблица POSITIONS

CREATE TABLE EMPLOYEES

(

id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

auth\_data INTEGER NOT NULL UNIQUE,

person\_id INTEGER NOT NULL UNIQUE,

position\_id INTEGER NOT NULL,

hire\_date DATE NOT NULL,

education NVARCHAR2(256) NOT NULL,

phone NVARCHAR2(20) NOT NULL,

salary NUMBER(10,2) NOT NULL,

on\_vacation CHAR(1) NOT NULL,

CONSTRAINT FK\_EMPLOYEE\_AUTHDATA

FOREIGN KEY (auth\_data)

REFERENCES USERS(id),

CONSTRAINT FK\_EMPLOYEE\_PERSON

FOREIGN KEY (person\_id)

REFERENCES PERSONS(id),

CONSTRAINT FK\_EMPLOYEE\_POSITION

FOREIGN KEY (position\_id)

REFERENCES POSITIONS(id),

CONSTRAINT CHK\_EMPLOYEE\_ONVACATION CHECK (on\_vacation in ('0', '1')) *-- 0 - not on vacation, 1 - on vacation*

) TABLESPACE TS\_MEDCENTER;

Листинг А.9 – Таблица EMPLOYEES

CREATE TABLE DEPARTMENTS

(

id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

department\_name NVARCHAR2(256) NOT NULL,

address\_id INTEGER NOT NULL,

department\_manager INTEGER, -- EMPLOYEE ID, NULL BECAUSE IT MAY BE BUILDING OR SMTH.

CONSTRAINT FK\_DEPARTMENT\_EMPLOYEE

FOREIGN KEY (department\_manager)

REFERENCES EMPLOYEES(id),

CONSTRAINT FK\_DEPARTMENT\_ADDRESS

FOREIGN KEY (address\_id)

REFERENCES ADDRESSES(id)

) TABLESPACE TS\_MEDCENTER;

Листинг А.10 – Таблица DEPARTMENTS

CREATE TABLE DEPARTMENT\_EMPLOYEE

(

id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

department\_id INTEGER NOT NULL,

employee\_id INTEGER NOT NULL,

CONSTRAINT FK\_DEPARTMENT\_EMPLOYEE\_DEPARTMENT

FOREIGN KEY (department\_id)

REFERENCES DEPARTMENTS(id),

CONSTRAINT FK\_DEPARTMENT\_EMPLOYEE\_EMPLOYEE

FOREIGN KEY (employee\_id)

REFERENCES EMPLOYEES(id)

);

Листинг А.11 – Таблица DEPARTMENT\_EMPLOYEE

CREATE TABLE TALONS

(

id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

talon\_date DATE NOT NULL,

employee\_id INTEGER NOT NULL,

patient\_id INTEGER,

CONSTRAINT FK\_TALON\_EMPLOYEE

FOREIGN KEY (employee\_id)

REFERENCES EMPLOYEES(id),

CONSTRAINT FK\_TALON\_PATIENT

FOREIGN KEY (patient\_id)

REFERENCES PATIENTS(id)

) TABLESPACE TS\_MEDCENTER;

Листинг А.12 – Таблица TALONS

CREATE TABLE PHARMACY

(

id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

drug NVARCHAR2(256) NOT NULL,

price NUMBER(10,2) NOT NULL,

stock INTEGER NOT NULL,

need\_recipe CHAR(1) NOT NULL,

supplier\_id INTEGER NOT NULL,

CONSTRAINT FK\_DRUG\_SUPPLIER

FOREIGN KEY (supplier\_id)

REFERENCES SUPPLIERS(id),

CONSTRAINT CHK\_NEED\_RECIPE CHECK (need\_recipe in ('0', '1')) *-- 0 - no need in recipe, 1 - need recipe*

) TABLESPACE TS\_MEDCENTER;

Листинг А.13 – Таблица PHARMACY

CREATE TABLE SUPPLIERS

(

id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

supplier\_name NVARCHAR2(256) NOT NULL,

supplier\_country NVARCHAR2(64) NOT NULL

) TABLESPACE TS\_MEDCENTER;

Листинг А.14 – Таблица SUPPLIERS

CREATE TABLE PRICELIST

(

id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

position\_id INTEGER NOT NULL,

service NVARCHAR2(256) NOT NULL,

price NUMBER(10, 2) NOT NULL,

CONSTRAINT FK\_PRICELIST\_POSITITON

FOREIGN KEY (position\_id)

REFERENCES POSITIONS(id)

) TABLESPACE TS\_MEDCENTER;

Листинг А.15 – Таблица PRICELIST

CREATE TABLE TREATMENTS

(

id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

employee\_id INTEGER NOT NULL,

patient\_id INTEGER NOT NULL,

start\_of\_treatment DATE NOT NULL,

end\_of\_treatment DATE,

diagnosis NVARCHAR2(256) NOT NULL,

treatment\_info NVARCHAR2(1024) NOT NULL,

recommendations NVARCHAR2(1024) NOT NULL,

CONSTRAINT FK\_TREATMENTS\_EMPLOYEE

FOREIGN KEY (employee\_id)

REFERENCES EMPLOYEES(id),

CONSTRAINT FK\_TREATMENTS\_PATIENT

FOREIGN KEY (patient\_id)

REFERENCES PATIENTS(id)

) TABLESPACE TS\_MEDCENTER;

Листинг А.16 – Таблица TREATMENTS

CREATE TABLE COMMENTS

(

id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

user\_id INTEGER NOT NULL,

employee\_id INTEGER NOT NULL,

comment\_text NVARCHAR2(1024) NOT NULL,

CONSTRAINT FK\_COMMENT\_USER

FOREIGN KEY (user\_id)

REFERENCES USERS(id),

CONSTRAINT FK\_COMMENT\_EMPLOYEE

FOREIGN KEY (employee\_id)

REFERENCES EMPLOYEES(id)

) TABLESPACE TS\_MEDCENTER;

Листинг А.17 – Таблица COMMENTS

CREATE TABLESPACE TS\_USERS

DATAFILE 'TS\_USERS.dbf'

SIZE 50M

AUTOEXTEND ON NEXT 10M

EXTENT MANAGEMENT LOCAL;

Листинг А.18 – Табличное пространство TS\_USERS

CREATE TABLESPACE TS\_MEDCENTER

DATAFILE 'TS\_MEDCENTER.dbf'

SIZE 50M

AUTOEXTEND ON NEXT 10M

EXTENT MANAGEMENT LOCAL;

Листинг А.19 – Табличное пространство TS\_MECENTER

# Приложение Б

Пакеты и процедуры

CREATE OR REPLACE PACKAGE ADDRESSES\_tapi

IS

PROCEDURE create\_address

(

p\_region in ADDRESSES.region%TYPE,

p\_town in ADDRESSES.town%TYPE,

p\_street in ADDRESSES.street%TYPE,

p\_house\_number in ADDRESSES.house\_number%TYPE,

p\_flat in ADDRESSES.flat%TYPE

);

PROCEDURE update\_address

(

p\_id in ADDRESSES.id%TYPE,

p\_region in ADDRESSES.region%TYPE,

p\_town in ADDRESSES.town%TYPE,

p\_street in ADDRESSES.street%TYPE,

p\_house\_number in ADDRESSES.house\_number%TYPE,

p\_flat in ADDRESSES.flat%TYPE

);

PROCEDURE delete\_address

(

p\_id in ADDRESSES.id%TYPE

);

END ADDRESSES\_tapi;

*--/*

*--/*

CREATE OR REPLACE PACKAGE BODY ADDRESSES\_tapi

IS

PROCEDURE create\_address

(

p\_region in ADDRESSES.region%TYPE,

p\_town in ADDRESSES.town%TYPE,

p\_street in ADDRESSES.street%TYPE,

p\_house\_number in ADDRESSES.house\_number%TYPE,

p\_flat in ADDRESSES.flat%TYPE

)

IS

empty\_parameter\_ex EXCEPTION;

BEGIN

IF TRIM(p\_region) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_town) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_street) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_house\_number) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_flat) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

INSERT INTO ADDRESSES(region, town, street, house\_number, flat)

VALUES (TRIM(p\_region), TRIM(p\_town), TRIM(p\_street), TRIM(p\_house\_number), TRIM(p\_flat));

COMMIT;

EXCEPTION

WHEN empty\_parameter\_ex THEN

dbms\_output.put\_line('Empty parameter');

END;

PROCEDURE update\_address

(

p\_id in ADDRESSES.id%TYPE,

p\_region in ADDRESSES.region%TYPE,

p\_town in ADDRESSES.town%TYPE,

p\_street in ADDRESSES.street%TYPE,

p\_house\_number in ADDRESSES.house\_number%TYPE,

p\_flat in ADDRESSES.flat%TYPE

)

IS

BEGIN

UPDATE ADDRESSES

SET region = nvl(TRIM(p\_region), region),

town = nvl(TRIM(p\_town), town),

street = nvl(TRIM(p\_street), street),

house\_number = nvl(TRIM(p\_house\_number), house\_number),

flat = nvl(TRIM(p\_flat), flat)

WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure\_error');

END;

PROCEDURE delete\_address

(

p\_id in ADDRESSES.id%TYPE

)

IS

BEGIN

DELETE FROM ADDRESSES WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure\_error');

END;

END;

Листинг Б.1 – Пакет процедур ADDRESSES\_tapi

CREATE OR REPLACE PACKAGE PASSPORTS\_tapi

is

PROCEDURE create\_passport

(

p\_passport\_number in PASSPORTS.passport\_number%TYPE,

p\_date\_of\_issue in VARCHAR2,

p\_date\_of\_expiry in VARCHAR2,

p\_authority in PASSPORTS.authority%TYPE

);

PROCEDURE update\_passport

(

p\_id in PASSPORTS.id%TYPE,

p\_passport\_number in PASSPORTS.passport\_number%TYPE,

p\_date\_of\_issue VARCHAR2,

p\_date\_of\_expiry VARCHAR2,

p\_authority in PASSPORTS.authority%TYPE

);

PROCEDURE delete\_passport

(

p\_id in PASSPORTS.id%TYPE

);

END PASSPORTS\_tapi;

--/

--/

CREATE OR REPLACE PACKAGE BODY PASSPORTS\_tapi

IS

PROCEDURE create\_passport

(

p\_passport\_number in PASSPORTS.passport\_number%TYPE,

p\_date\_of\_issue in VARCHAR2,

p\_date\_of\_expiry in VARCHAR2,

p\_authority in PASSPORTS.authority%TYPE

)

IS

empty\_parameter\_ex EXCEPTION;

BEGIN

IF TRIM(p\_passport\_number) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_date\_of\_issue) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_date\_of\_expiry) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_authority) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

INSERT INTO PASSPORTS(passport\_number, date\_of\_issue, date\_of\_expiry, authority)

VALUES (p\_passport\_number, to\_date(p\_date\_of\_issue, 'dd.mm.yyyy'), to\_date(p\_date\_of\_expiry, 'dd.mm.yyyy'), p\_authority);

COMMIT;

EXCEPTION

WHEN empty\_parameter\_ex THEN

dbms\_output.put\_line('Empty parameter');

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE update\_passport

(

p\_id in PASSPORTS.id%TYPE,

p\_passport\_number in PASSPORTS.passport\_number%TYPE,

p\_date\_of\_issue VARCHAR2,

p\_date\_of\_expiry VARCHAR2,

p\_authority in PASSPORTS.authority%TYPE

)

IS

BEGIN

UPDATE PASSPORTS

SET passport\_number = nvl(TRIM(p\_passport\_number), passport\_number),

date\_of\_issue = nvl(to\_date(TRIM(p\_date\_of\_issue), 'dd.mm.yyyy'), date\_of\_issue),

date\_of\_expiry = nvl(to\_date(TRIM(p\_date\_of\_expiry), 'dd.mm.yyyy'), date\_of\_expiry),

authority = nvl(TRIM(p\_authority), authority)

WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE delete\_passport

(

p\_id in PASSPORTS.id%TYPE

)

IS

BEGIN

DELETE FROM PASSPORTS WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error!');

END;

END;

Листинг Б.2 – Пакет процедур PASSPRTS\_tapi

create or replace package USERS\_tapi

is

PROCEDURE create\_user

(

p\_user\_role in USERS.user\_role%TYPE,

p\_email in USERS.email%TYPE,

p\_password in VARCHAR2

);

PROCEDURE update\_user

(

p\_id in USERS.id%TYPE,

p\_user\_role in USERS.user\_role%TYPE,

p\_email in USERS.email%TYPE,

p\_password in VARCHAR2

);

PROCEDURE delete\_user

(

p\_id in USERS.id%TYPE

);

end USERS\_tapi;

*--/*

*--/*

create or replace package body USERS\_tapi

is

PROCEDURE create\_user

(

p\_user\_role in USERS.user\_role%TYPE,

p\_email in USERS.email%TYPE,

p\_password in VARCHAR2

)

IS

empty\_parameter\_ex EXCEPTION;

BEGIN

IF TRIM(p\_email) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_password) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

INSERT INTO USERS(user\_role, email, password)

VALUES (p\_user\_role, TRIM(p\_email), (select hash\_password(TRIM(p\_password)) from DUAL));

COMMIT;

EXCEPTION

WHEN empty\_parameter\_ex THEN

dbms\_output.put\_line('Empty parameter');

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error');

END;

PROCEDURE update\_user

(

p\_id in USERS.id%TYPE,

p\_user\_role in USERS.user\_role%TYPE,

p\_email in USERS.email%TYPE,

p\_password in VARCHAR2

)

IS

BEGIN

UPDATE USERS

SET user\_role = NVL(TRIM(p\_user\_role), user\_role),

email = NVL(TRIM(p\_email), email),

password = NVL(hash\_password(TRIM(p\_password)),password)

WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error');

END;

PROCEDURE delete\_user

(

p\_id in USERS.id%TYPE

)

IS

BEGIN

DELETE FROM USERS WHERE id = p\_id;

COMMIT;

END;

END;

Листинг Б.3 – Пакет процедур USERS\_tapi

create or replace package PERSONS\_tapi

is

PROCEDURE create\_person

(

p\_first\_name in PERSONS.first\_name%TYPE,

p\_second\_name in PERSONS.second\_name%TYPE,

p\_last\_name in PERSONS.last\_name%TYPE,

p\_passport\_id in PERSONS.passport\_id%TYPE,

p\_birth\_date VARCHAR2,

p\_gender in PERSONS.gender%TYPE

);

PROCEDURE update\_person

(

p\_id in PERSONS.id%TYPE,

p\_first\_name in PERSONS.first\_name%TYPE,

p\_second\_name in PERSONS.second\_name%TYPE,

p\_last\_name in PERSONS.last\_name%TYPE,

p\_passport\_id in PERSONS.passport\_id%TYPE,

p\_birth\_date VARCHAR2,

p\_gender in PERSONS.gender%TYPE

);

PROCEDURE delete\_person

(

p\_id in PERSONS.id%TYPE

);

END PERSONS\_tapi;

*--/*

*--/*

create or replace package body PERSONS\_tapi

is

PROCEDURE create\_person

(

p\_first\_name in PERSONS.first\_name%TYPE,

p\_second\_name in PERSONS.second\_name%TYPE,

p\_last\_name in PERSONS.last\_name%TYPE,

p\_passport\_id in PERSONS.passport\_id%TYPE,

p\_birth\_date VARCHAR2,

p\_gender in PERSONS.gender%TYPE

)

IS

empty\_parameter\_ex EXCEPTION;

BEGIN

IF TRIM(p\_first\_name) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_second\_name) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_last\_name) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_birth\_date) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_gender) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

INSERT INTO PERSONS(first\_name, second\_name, last\_name, passport\_id, birth\_date, gender)

VALUES (TRIM(p\_first\_name), TRIM(p\_second\_name), TRIM(p\_last\_name), p\_passport\_id, to\_date(TRIM(p\_birth\_date), 'dd.mm.yyyy'), TRIM(p\_gender));

COMMIT;

EXCEPTION

WHEN empty\_parameter\_ex THEN

dbms\_output.put\_line('Empty parameter');

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE update\_person

(

p\_id in PERSONS.id%TYPE,

p\_first\_name in PERSONS.first\_name%TYPE,

p\_second\_name in PERSONS.second\_name%TYPE,

p\_last\_name in PERSONS.last\_name%TYPE,

p\_passport\_id in PERSONS.passport\_id%TYPE,

p\_birth\_date VARCHAR2,

p\_gender in PERSONS.gender%TYPE

)

IS

BEGIN

UPDATE PERSONS

SET first\_name = nvl(TRIM(p\_first\_name), first\_name),

second\_name = nvl(TRIM(p\_second\_name), second\_name),

last\_name = nvl(TRIM(p\_last\_name), last\_name),

passport\_id = nvl(p\_passport\_id, passport\_id),

birth\_date = nvl(to\_date(TRIM(p\_birth\_date), 'dd.mm.yyyy'), birth\_date),

gender = nvl(TRIM(p\_gender), gender)

WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE delete\_person

(

p\_id in PERSONS.id%TYPE

)

IS

BEGIN

DELETE FROM PERSONS WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

END;

Листинг Б.4 – Пакет процедур USERS\_tapi

create or replace package PERSON\_ADDRESS\_tapi

is

PROCEDURE create\_person\_address

(

p\_person\_id in PERSON\_ADDRESS.person\_id%TYPE,

p\_address\_id in PERSON\_ADDRESS.address\_id%TYPE

);

PROCEDURE update\_person\_address

(

p\_id in PERSON\_ADDRESS.id%TYPE,

p\_person\_id in PERSON\_ADDRESS.person\_id%TYPE,

p\_address\_id in PERSON\_ADDRESS.address\_id%TYPE

);

PROCEDURE delete\_person\_address

(

p\_id in PERSON\_ADDRESS.id%TYPE

);

END PERSON\_ADDRESS\_tapi;

*--/*

*--/*

create or replace package body PERSON\_ADDRESS\_tapi

is

PROCEDURE create\_person\_address

(

p\_person\_id in PERSON\_ADDRESS.person\_id%TYPE,

p\_address\_id in PERSON\_ADDRESS.address\_id%TYPE

)

IS

empty\_parameter\_ex EXCEPTION;

BEGIN

IF p\_person\_id IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF p\_address\_id IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

INSERT INTO PERSON\_ADDRESS(person\_id, address\_id) VALUES (p\_person\_id, p\_address\_id);

COMMIT;

EXCEPTION

WHEN empty\_parameter\_ex THEN

dbms\_output.put\_line('Empty parameter');

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE update\_person\_address

(

p\_id in PERSON\_ADDRESS.id%TYPE,

p\_person\_id in PERSON\_ADDRESS.person\_id%TYPE,

p\_address\_id in PERSON\_ADDRESS.address\_id%TYPE

)

IS

BEGIN

UPDATE PERSON\_ADDRESS

SET person\_id = nvl(p\_person\_id, person\_id),

address\_id = nvl(p\_address\_id, address\_id)

WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE delete\_person\_address

(

p\_id in PERSON\_ADDRESS.id%TYPE

)

IS

BEGIN

DELETE FROM PERSON\_ADDRESS WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

END;

Листинг Б.5 – Пакет процедур PERSON\_ADDRESS\_tapi

create or replace package PATIENTS\_tapi

is

PROCEDURE create\_patient

(

p\_auth\_data in PATIENTS.auth\_data%TYPE,

p\_person\_id in PATIENTS.person\_id%TYPE,

p\_phone in PATIENTS.phone%TYPE

);

PROCEDURE update\_patient

(

p\_id in PATIENTS.id%TYPE,

p\_auth\_data in PATIENTS.auth\_data%TYPE,

p\_person\_id in PATIENTS.person\_id%TYPE,

p\_phone in PATIENTS.phone%TYPE

);

PROCEDURE delete\_patient

(

p\_id in PATIENTS.id%TYPE

);

END PATIENTS\_tapi;

*--/*

*--/*

create or replace package body PATIENTS\_tapi

is

PROCEDURE create\_patient

(

p\_auth\_data in PATIENTS.auth\_data%TYPE,

p\_person\_id in PATIENTS.person\_id%TYPE,

p\_phone in PATIENTS.phone%TYPE

)

IS

empty\_parameter\_ex EXCEPTION;

BEGIN

IF p\_auth\_data IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF p\_person\_id IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_phone) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

INSERT INTO PATIENTS(auth\_data, person\_id, phone) VALUES (p\_auth\_data, p\_person\_id, TRIM(p\_phone));

COMMIT;

EXCEPTION

WHEN empty\_parameter\_ex THEN

dbms\_output.put\_line('Empty parameter');

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE update\_patient

(

p\_id in PATIENTS.id%TYPE,

p\_auth\_data in PATIENTS.auth\_data%TYPE,

p\_person\_id in PATIENTS.person\_id%TYPE,

p\_phone in PATIENTS.phone%TYPE

)

IS

BEGIN

UPDATE PATIENTS

SET auth\_data = nvl(p\_auth\_data, auth\_data),

person\_id = nvl(p\_person\_id, person\_id),

phone = nvl(TRIM(p\_phone), phone)

WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE delete\_patient

(

p\_id in PATIENTS.id%TYPE

)

IS

BEGIN

DELETE FROM PATIENTS WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

END;

Листинг Б.6 – Пакет процедур PATIENTS\_tapi

create or replace package EMPLOYEES\_tapi

is

PROCEDURE create\_employee

(

p\_auth\_data in EMPLOYEES.auth\_data%TYPE,

p\_person\_id in EMPLOYEES.person\_id%TYPE,

p\_position\_id in EMPLOYEES.position\_id%TYPE,

p\_hire\_date in VARCHAR2,

p\_education in EMPLOYEES.education%TYPE,

p\_phone in EMPLOYEES.phone%TYPE,

p\_salary in EMPLOYEES.salary%TYPE,

p\_on\_vacation in EMPLOYEES.on\_vacation%TYPE

);

PROCEDURE update\_employee

(

p\_id in EMPLOYEES.id%TYPE,

p\_auth\_data in EMPLOYEES.auth\_data%TYPE,

p\_person\_id in EMPLOYEES.person\_id%TYPE,

p\_position\_id in EMPLOYEES.position\_id%TYPE,

p\_hire\_date in VARCHAR2,

p\_education in EMPLOYEES.education%TYPE,

p\_phone in EMPLOYEES.phone%TYPE,

p\_salary in EMPLOYEES.salary%TYPE,

p\_on\_vacation in EMPLOYEES.on\_vacation%TYPE

);

PROCEDURE delete\_employee

(

p\_id in EMPLOYEES.id%TYPE

);

END EMPLOYEES\_tapi;

*--/*

*--/*

create or replace package body EMPLOYEES\_tapi

is

PROCEDURE create\_employee

(

p\_auth\_data in EMPLOYEES.auth\_data%TYPE,

p\_person\_id in EMPLOYEES.person\_id%TYPE,

p\_position\_id in EMPLOYEES.position\_id%TYPE,

p\_hire\_date in VARCHAR2,

p\_education in EMPLOYEES.education%TYPE,

p\_phone in EMPLOYEES.phone%TYPE,

p\_salary in EMPLOYEES.salary%TYPE,

p\_on\_vacation in EMPLOYEES.on\_vacation%TYPE

)

IS

empty\_parameter\_ex EXCEPTION;

BEGIN

IF p\_auth\_data IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF p\_person\_id IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF p\_position\_id IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF p\_hire\_date IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF p\_education IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF p\_salary IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_phone) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF p\_on\_vacation IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

INSERT INTO EMPLOYEES(auth\_data, person\_id, position\_id, hire\_date, education, phone, salary, on\_vacation)

VALUES(p\_auth\_data, p\_person\_id, p\_position\_id, to\_date(TRIM(p\_hire\_date), 'dd.mm.yyyy'), TRIM(p\_education), TRIM(p\_phone), p\_salary, TRIM(p\_on\_vacation));

COMMIT;

EXCEPTION

WHEN empty\_parameter\_ex THEN

dbms\_output.put\_line('Empty parameter');

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE update\_employee

(

p\_id in EMPLOYEES.id%TYPE,

p\_auth\_data in EMPLOYEES.auth\_data%TYPE,

p\_person\_id in EMPLOYEES.person\_id%TYPE,

p\_position\_id in EMPLOYEES.position\_id%TYPE,

p\_hire\_date in VARCHAR2,

p\_education in EMPLOYEES.education%TYPE,

p\_phone in EMPLOYEES.phone%TYPE,

p\_salary in EMPLOYEES.salary%TYPE,

p\_on\_vacation in EMPLOYEES.on\_vacation%TYPE

)

IS

BEGIN

UPDATE EMPLOYEES

SET auth\_data = nvl(p\_auth\_data, auth\_data),

person\_id = nvl(p\_person\_id, person\_id),

position\_id = nvl(p\_position\_id, position\_id),

hire\_date = nvl(to\_date(TRIM(p\_hire\_date), 'dd.mm.yyyy'), hire\_date),

education = nvl(p\_education, education),

phone = nvl(p\_phone, phone),

salary = nvl(p\_salary, salary),

on\_vacation = nvl(TRIM(p\_on\_vacation), on\_vacation)

WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE delete\_employee

(

p\_id in EMPLOYEES.id%TYPE

)

IS

BEGIN

DELETE FROM EMPLOYEES WHERE id = p\_id;

COMMIT;

END;

END;

Листинг Б.7 – Пакет процедур EMPLOYEES\_tapi

create or replace package DEPARTMENTS\_tapi

is

PROCEDURE create\_department

(

p\_dep\_name in DEPARTMENTS.department\_name%TYPE,

p\_dep\_manager in DEPARTMENTS.department\_manager%TYPE

);

PROCEDURE update\_department

(

p\_id in DEPARTMENTS.id%TYPE,

p\_dep\_name in DEPARTMENTS.department\_name%TYPE,

p\_dep\_manager in DEPARTMENTS.department\_manager%TYPE

);

PROCEDURE delete\_department

(

p\_id in DEPARTMENTS.id%TYPE

);

END DEPARTMENTS\_tapi;

*--/*

*--/*

create or replace package body DEPARTMENTS\_tapi

is

PROCEDURE create\_department

(

p\_dep\_name in DEPARTMENTS.department\_name%TYPE,

p\_dep\_manager in DEPARTMENTS.department\_manager%TYPE

)

IS

empty\_parameter\_ex EXCEPTION;

BEGIN

IF p\_dep\_name IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

INSERT INTO DEPARTMENTS(department\_name, department\_manager) VALUES (TRIM(p\_dep\_name), p\_dep\_manager);

COMMIT;

EXCEPTION

WHEN empty\_parameter\_ex THEN

dbms\_output.put\_line('Empty parameter');

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE update\_department

(

p\_id in DEPARTMENTS.id%TYPE,

p\_dep\_name in DEPARTMENTS.department\_name%TYPE,

p\_dep\_manager in DEPARTMENTS.department\_manager%TYPE

)

IS

BEGIN

UPDATE DEPARTMENTS

SET department\_name = nvl(TRIM(p\_dep\_name), department\_name),

department\_manager = nvl(p\_dep\_manager, department\_manager)

WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE delete\_department

(

p\_id in DEPARTMENTS.id%TYPE

)

IS

BEGIN

DELETE FROM DEPARTMENTS WHERE id = p\_id;

COMMIT;

END;

END;

Листинг Б.8 – Пакет процедур DEPARTMENTS\_tapi

create or replace package POSITIONS\_tapi

is

PROCEDURE create\_position

(

p\_pos\_name in POSITIONS.position\_name%TYPE,

p\_pos\_type in POSITIONS.position\_type%TYPE

);

PROCEDURE update\_position

(

p\_id in POSITIONS.id%TYPE,

p\_pos\_name in POSITIONS.position\_name%TYPE,

p\_pos\_type in POSITIONS.position\_type%TYPE

);

PROCEDURE delete\_position

(

p\_id in POSITIONS.id%TYPE

);

END POSITIONS\_tapi;

*--/*

*--/*

create or replace package body POSITIONS\_tapi

is

PROCEDURE create\_position

(

p\_pos\_name in POSITIONS.position\_name%TYPE,

p\_pos\_type in POSITIONS.position\_type%TYPE

)

IS

empty\_parameter\_ex EXCEPTION;

BEGIN

IF p\_pos\_name IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_pos\_type) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

INSERT INTO POSITIONS(position\_name, position\_type)

VALUES (TRIM(p\_pos\_name), p\_pos\_type);

COMMIT;

EXCEPTION

WHEN empty\_parameter\_ex THEN

dbms\_output.put\_line('Empty parameter');

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE update\_position

(

p\_id in POSITIONS.id%TYPE,

p\_pos\_name in POSITIONS.position\_name%TYPE,

p\_pos\_type in POSITIONS.position\_type%TYPE

)

IS

BEGIN

UPDATE POSITIONS

SET position\_name = nvl(TRIM(p\_pos\_name), position\_name),

position\_type = nvl(TRIM(p\_pos\_type), position\_type)

WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE delete\_position

(

p\_id in POSITIONS.id%TYPE

)

IS

BEGIN

DELETE FROM POSITIONS WHERE id = p\_id;

COMMIT;

END;

END;

Листинг Б.9 – Пакет процедур POSITIONS\_tapi

create or replace package DEPARTMENT\_EMPLOYEE\_tapi

is

PROCEDURE create\_department\_employee

(

p\_dep\_id in DEPARTMENT\_EMPLOYEE.department\_id%TYPE,

p\_emp\_id in DEPARTMENT\_EMPLOYEE.employee\_id%TYPE

);

PROCEDURE update\_branch\_department

(

p\_id in DEPARTMENT\_EMPLOYEE.id%TYPE,

p\_dep\_id in DEPARTMENT\_EMPLOYEE.department\_id%TYPE,

p\_emp\_id in DEPARTMENT\_EMPLOYEE.employee\_id%TYPE

);

PROCEDURE delete\_department\_employee

(

p\_id in DEPARTMENT\_EMPLOYEE.id%TYPE

);

END DEPARTMENT\_EMPLOYEE\_tapi;

*--/*

*--/*

create or replace package body DEPARTMENT\_EMPLOYEE\_tapi

is

PROCEDURE create\_department\_employee

(

p\_dep\_id in DEPARTMENT\_EMPLOYEE.department\_id%TYPE,

p\_emp\_id in DEPARTMENT\_EMPLOYEE.employee\_id%TYPE

)

IS

empty\_parameter\_ex EXCEPTION;

BEGIN

IF p\_dep\_id IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF p\_emp\_id IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

INSERT INTO DEPARTMENT\_EMPLOYEE(department\_id, employee\_id) VALUES (p\_dep\_id, p\_emp\_id);

COMMIT;

EXCEPTION

WHEN empty\_parameter\_ex THEN

dbms\_output.put\_line('Empty parameter');

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE update\_branch\_department

(

p\_id in DEPARTMENT\_EMPLOYEE.id%TYPE,

p\_dep\_id in DEPARTMENT\_EMPLOYEE.department\_id%TYPE,

p\_emp\_id in DEPARTMENT\_EMPLOYEE.employee\_id%TYPE

)

IS

BEGIN

UPDATE DEPARTMENT\_EMPLOYEE

SET employee\_id = nvl(p\_emp\_id, employee\_id),

department\_id = nvl(p\_dep\_id, department\_id)

WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE delete\_department\_employee

(

p\_id in DEPARTMENT\_EMPLOYEE.id%TYPE

)

IS

BEGIN

DELETE FROM DEPARTMENT\_EMPLOYEE WHERE id = p\_id;

COMMIT;

END;

END;

Листинг Б.10 – Пакет процедур DEPARTMENT\_EMPLOYEE\_tapi

create or replace package TALONS\_tapi

is

PROCEDURE create\_talon

(

p\_t\_date in VARCHAR2,

p\_emp\_id in TALONS.employee\_id%TYPE

);

PROCEDURE update\_talon

(

p\_id in TALONS.id%TYPE,

p\_t\_date in VARCHAR2,

p\_emp\_id in TALONS.employee\_id%TYPE,

p\_patient\_id in TALONS.patient\_id%TYPE

);

PROCEDURE delete\_talon

(

p\_id in TALONS.id%TYPE

);

END TALONS\_tapi;

*--/*

*--/*

create or replace package body TALONS\_tapi

is

PROCEDURE create\_talon

(

p\_t\_date in VARCHAR2,

p\_emp\_id in TALONS.employee\_id%TYPE

)

IS

empty\_parameter\_ex EXCEPTION;

BEGIN

IF p\_t\_date IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF p\_emp\_id IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

INSERT INTO TALONS(talon\_date, employee\_id, patient\_id)

VALUES (to\_date(TRIM(p\_t\_date), 'DD.MM.YYYY HH24:MI'), p\_emp\_id, null);

COMMIT;

EXCEPTION

WHEN empty\_parameter\_ex THEN

dbms\_output.put\_line('Empty parameter');

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE update\_talon

(

p\_id in TALONS.id%TYPE,

p\_t\_date in VARCHAR2,

p\_emp\_id in TALONS.employee\_id%TYPE,

p\_patient\_id in TALONS.patient\_id%TYPE

)

IS

BEGIN

UPDATE TALONS

SET talon\_date = nvl(to\_date(TRIM(p\_t\_date), 'dd.mm.yyyy HH24:MI'), talon\_date),

employee\_id = nvl(p\_emp\_id, employee\_id),

patient\_id = nvl(p\_patient\_id, patient\_id)

WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE delete\_talon

(

p\_id in TALONS.id%TYPE

)

IS

BEGIN

DELETE FROM TALONS WHERE id = p\_id;

COMMIT;

END;

END;

Листинг Б.11 – Пакет процедур TALONS\_tapi

create or replace package SUPPLIERS\_tapi

is

PROCEDURE create\_supplier

(

p\_supplier\_name in SUPPLIERS.supplier\_name%TYPE,

p\_supplier\_country in SUPPLIERS.supplier\_country%TYPE

);

PROCEDURE update\_supplier

(

p\_id in SUPPLIERS.id%TYPE,

p\_supplier\_name in SUPPLIERS.supplier\_name%TYPE,

p\_supplier\_country in SUPPLIERS.supplier\_country%TYPE

);

PROCEDURE delete\_supplier

(

p\_id in SUPPLIERS.id%TYPE

);

END SUPPLIERS\_tapi;

*--/*

*--/*

create or replace package body SUPPLIERS\_tapi

is

PROCEDURE create\_supplier

(

p\_supplier\_name in SUPPLIERS.supplier\_name%TYPE,

p\_supplier\_country in SUPPLIERS.supplier\_country%TYPE

)

IS

empty\_parameter\_ex EXCEPTION;

BEGIN

IF TRIM(p\_supplier\_name) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_supplier\_country) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

INSERT INTO SUPPLIERS(supplier\_name, supplier\_country)

VALUES (TRIM(p\_supplier\_name), TRIM(p\_supplier\_country));

COMMIT;

EXCEPTION

WHEN empty\_parameter\_ex THEN

dbms\_output.put\_line('Empty parameter');

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE update\_supplier

(

p\_id in SUPPLIERS.id%TYPE,

p\_supplier\_name in SUPPLIERS.supplier\_name%TYPE,

p\_supplier\_country in SUPPLIERS.supplier\_country%TYPE

)

IS

BEGIN

UPDATE SUPPLIERS

SET supplier\_name = nvl(TRIM(p\_supplier\_name), supplier\_name),

supplier\_country = nvl(TRIM(p\_supplier\_country), supplier\_country)

WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE delete\_supplier

(

p\_id in SUPPLIERS.id%TYPE

)

IS

BEGIN

DELETE FROM SUPPLIERS WHERE id = p\_id;

COMMIT;

END;

END;

Листинг Б.12 – Пакет процедур SUPPLIERS\_tapi

create or replace package PHARMACY\_tapi

is

PROCEDURE create\_drug

(

p\_drug in PHARMACY.drug%TYPE,

p\_price in PHARMACY.price%TYPE,

p\_stock in PHARMACY.stock%TYPE,

p\_need\_recipe in PHARMACY.need\_recipe%TYPE,

p\_supplier\_id in PHARMACY.supplier\_id%TYPE

);

PROCEDURE update\_drug

(

p\_id in PHARMACY.id%TYPE,

p\_drug in PHARMACY.drug%TYPE,

p\_price in PHARMACY.price%TYPE,

p\_stock in PHARMACY.stock%TYPE,

p\_need\_recipe in PHARMACY.need\_recipe%TYPE,

p\_supplier\_id in PHARMACY.supplier\_id%TYPE

);

PROCEDURE delete\_drug

(

p\_id in PHARMACY.id%TYPE

);

END PHARMACY\_tapi;

*--/*

*--/*

create or replace package body PHARMACY\_tapi

is

PROCEDURE create\_drug

(

p\_drug in PHARMACY.drug%TYPE,

p\_price in PHARMACY.price%TYPE,

p\_stock in PHARMACY.stock%TYPE,

p\_need\_recipe in PHARMACY.need\_recipe%TYPE,

p\_supplier\_id in PHARMACY.supplier\_id%TYPE

)

IS

empty\_parameter\_ex EXCEPTION;

BEGIN

IF TRIM(p\_drug) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF p\_price IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF p\_stock IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_need\_recipe) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF p\_supplier\_id IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

INSERT INTO PHARMACY(drug, price, stock, need\_recipe, supplier\_id)

VALUES (TRIM(p\_drug), p\_price, p\_stock, TRIM(p\_need\_recipe), p\_supplier\_id);

COMMIT;

EXCEPTION

WHEN empty\_parameter\_ex THEN

dbms\_output.put\_line('Empty parameter');

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE update\_drug

(

p\_id in PHARMACY.id%TYPE,

p\_drug in PHARMACY.drug%TYPE,

p\_price in PHARMACY.price%TYPE,

p\_stock in PHARMACY.stock%TYPE,

p\_need\_recipe in PHARMACY.need\_recipe%TYPE,

p\_supplier\_id in PHARMACY.supplier\_id%TYPE

)

IS

BEGIN

UPDATE PHARMACY

SET drug = nvl(TRIM(p\_drug), drug),

price = nvl(p\_price, price),

stock = nvl(p\_stock, stock),

need\_recipe = nvl(TRIM(p\_need\_recipe), need\_recipe),

supplier\_id = nvl(p\_supplier\_id, supplier\_id)

WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE delete\_drug

(

p\_id in PHARMACY.id%TYPE

)

IS

BEGIN

DELETE FROM PHARMACY WHERE id = p\_id;

COMMIT;

END;

END;

Листинг Б.13 – Пакет процедур PHARMACY\_tapi

create or replace package COMMENTS\_tapi

is

PROCEDURE create\_comment

(

p\_user\_id in COMMENTS.user\_id%TYPE,

p\_employee\_id in COMMENTS.employee\_id%TYPE,

p\_comment\_text in COMMENTS.comment\_text%TYPE

);

PROCEDURE update\_comment

(

p\_id in COMMENTS.id%TYPE,

p\_user\_id in COMMENTS.user\_id%TYPE,

p\_employee\_id in COMMENTS.employee\_id%TYPE,

p\_comment\_text in COMMENTS.comment\_text%TYPE

);

PROCEDURE delete\_comment

(

p\_id in COMMENTS.id%TYPE

);

END COMMENTS\_tapi;

*--/*

*--/*

create or replace package body COMMENTS\_tapi

is

PROCEDURE create\_comment

(

p\_user\_id in COMMENTS.user\_id%TYPE,

p\_employee\_id in COMMENTS.employee\_id%TYPE,

p\_comment\_text in COMMENTS.comment\_text%TYPE

)

IS

empty\_parameter\_ex EXCEPTION;

BEGIN

IF p\_user\_id IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF p\_employee\_id IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_comment\_text) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

INSERT INTO COMMENTS(user\_id, employee\_id, comment\_text)

VALUES (p\_user\_id, p\_employee\_id, TRIM(p\_comment\_text));

COMMIT;

EXCEPTION

WHEN empty\_parameter\_ex THEN

dbms\_output.put\_line('Empty parameter');

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE update\_comment

(

p\_id in COMMENTS.id%TYPE,

p\_user\_id in COMMENTS.user\_id%TYPE,

p\_employee\_id in COMMENTS.employee\_id%TYPE,

p\_comment\_text in COMMENTS.comment\_text%TYPE

)

IS

BEGIN

UPDATE COMMENTS

SET user\_id = nvl(p\_user\_id, user\_id),

employee\_id = nvl(p\_employee\_id, employee\_id),

comment\_text = nvl(TRIM(p\_comment\_text), comment\_text)

WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE delete\_comment

(

p\_id in COMMENTS.id%TYPE

)

IS

BEGIN

DELETE FROM COMMENTS WHERE id = p\_id;

COMMIT;

END;

END;

Листинг Б.14 – Пакет процедур COMMENTS\_tapi

create or replace package PRICELIST\_tapi

is

PROCEDURE create\_listitem

(

p\_pos\_id in PRICELIST.position\_id%TYPE,

p\_service in PRICELIST.service%TYPE,

p\_price in PRICELIST.price%TYPE

);

PROCEDURE update\_listitem

(

p\_id in PHARMACY.id%TYPE,

p\_pos\_id in PRICELIST.position\_id%TYPE,

p\_service in PRICELIST.service%TYPE,

p\_price in PRICELIST.price%TYPE

);

PROCEDURE delete\_listitem

(

p\_id in PRICELIST.id%TYPE

);

END PRICELIST\_tapi;

*--/*

*--/*

create or replace package body PRICELIST\_tapi

is

PROCEDURE create\_listitem

(

p\_pos\_id in PRICELIST.position\_id%TYPE,

p\_service in PRICELIST.service%TYPE,

p\_price in PRICELIST.price%TYPE

)

IS

empty\_parameter\_ex EXCEPTION;

BEGIN

IF p\_pos\_id IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_service) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF p\_price IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

INSERT INTO PRICELIST(position\_id, service, price)

VALUES (p\_pos\_id, TRIM(p\_service), p\_price);

COMMIT;

EXCEPTION

WHEN empty\_parameter\_ex THEN

dbms\_output.put\_line('Empty parameter');

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE update\_listitem

(

p\_id in PHARMACY.id%TYPE,

p\_pos\_id in PRICELIST.position\_id%TYPE,

p\_service in PRICELIST.service%TYPE,

p\_price in PRICELIST.price%TYPE

)

IS

BEGIN

UPDATE PRICELIST

SET position\_id = nvl(p\_pos\_id, position\_id),

service = nvl(TRIM(p\_service), service),

price = nvl(p\_price, price)

WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE delete\_listitem

(

p\_id in PRICELIST.id%TYPE

)

IS

BEGIN

DELETE FROM PRICELIST WHERE id = p\_id;

COMMIT;

END;

END;

Листинг Б.16 – Пакет процедур PRICELIST\_tapi

create or replace package TREATMENTS\_tapi

is

PROCEDURE create\_treatment

(

p\_emp\_id in TREATMENTS.employee\_id%TYPE,

p\_patient\_id in TREATMENTS.patient\_id%TYPE,

p\_start in VARCHAR2,

p\_diagnosis in TREATMENTS.diagnosis%TYPE,

p\_info in TREATMENTS.treatment\_info%TYPE,

p\_recomms in TREATMENTS.recommendations%TYPE

);

PROCEDURE update\_treatment

(

p\_id in TREATMENTS.id%TYPE,

p\_emp\_id in TREATMENTS.employee\_id%TYPE,

p\_patient\_id in TREATMENTS.patient\_id%TYPE,

p\_start in VARCHAR2,

p\_end in VARCHAR2,

p\_diagnosis in TREATMENTS.diagnosis%TYPE,

p\_info in TREATMENTS.treatment\_info%TYPE,

p\_recomms in TREATMENTS.recommendations%TYPE

);

PROCEDURE delete\_treatment

(

p\_id in TREATMENTS.id%TYPE

);

END TREATMENTS\_tapi;

*--/*

*--/*

create or replace package body TREATMENTS\_tapi

is

PROCEDURE create\_treatment

(

p\_emp\_id in TREATMENTS.employee\_id%TYPE,

p\_patient\_id in TREATMENTS.patient\_id%TYPE,

p\_start in VARCHAR2,

p\_diagnosis in TREATMENTS.diagnosis%TYPE,

p\_info in TREATMENTS.treatment\_info%TYPE,

p\_recomms in TREATMENTS.recommendations%TYPE

)

IS

empty\_parameter\_ex EXCEPTION;

BEGIN

IF p\_emp\_id IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF p\_patient\_id IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_start) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_diagnosis) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_info) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

IF TRIM(p\_recomms) IS NULL THEN

RAISE empty\_parameter\_ex;

END IF;

INSERT INTO TREATMENTS(employee\_id, patient\_id, start\_of\_treatment, end\_of\_treatment, diagnosis, treatment\_info, recommendations)

VALUES (p\_emp\_id, p\_patient\_id, to\_date(TRIM(p\_start), 'dd.mm.yyyy hh24:mi'), null, TRIM(p\_diagnosis), TRIM(p\_info), TRIM(p\_recomms));

COMMIT;

EXCEPTION

WHEN empty\_parameter\_ex THEN

dbms\_output.put\_line('Empty parameter');

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE update\_treatment

(

p\_id in TREATMENTS.id%TYPE,

p\_emp\_id in TREATMENTS.employee\_id%TYPE,

p\_patient\_id in TREATMENTS.patient\_id%TYPE,

p\_start in VARCHAR2,

p\_end in VARCHAR2,

p\_diagnosis in TREATMENTS.diagnosis%TYPE,

p\_info in TREATMENTS.treatment\_info%TYPE,

p\_recomms in TREATMENTS.recommendations%TYPE

)

IS

BEGIN

UPDATE TREATMENTS

SET employee\_id = nvl(p\_emp\_id, employee\_id),

patient\_id = nvl(p\_patient\_id, patient\_id),

start\_of\_treatment = nvl(to\_date(TRIM(p\_start), 'dd.mm.yyyy hh24:mi'), start\_of\_treatment),

end\_of\_treatment = nvl(to\_date(TRIM(p\_end), 'dd.mm.yyyy hh24:mi'), end\_of\_treatment),

diagnosis = nvl(TRIM(p\_diagnosis), diagnosis),

treatment\_info = nvl(TRIM(p\_info), treatment\_info),

recommendations = nvl(TRIM(p\_recomms), recommendations)

WHERE id = p\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Procedure error! Check you parameters');

END;

PROCEDURE delete\_treatment

(

p\_id in TREATMENTS.id%TYPE

)

IS

BEGIN

DELETE FROM TREATMENTS WHERE id = p\_id;

COMMIT;

END;

END;

Листинг Б.17 – Пакет процедур TREATMENTS\_tapi