

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

подпись, дата

Д. А. Воликов

Руководитель

Ассистент

должность, учен. степень, ученое звание

подпись, дата

Кантарович В. С.

Допущен к защите _____

дата, подпись

Курсовой проект защищен с оценкой _____

Руководитель _____

подпись

дата

Кантарович В. С.

инициалы и фамилия

Минск 2023

Содержание

Введение.....	6
1. Анализ требований к программному средству	7
1.1. Аналитический обзор аналогов.....	7
1.1.1. Аналог «Talon.by».....	7
1.1.2. Аналог «Городская студенческая поликлиника»	7
1.1.3. Аналог «3-я центральная районная клиническая поликлиника»	8
1.2. Разработка функциональных требований, определение вариантов использования.....	9
1.2.1. Разработка требований к базе данных	9
1.2.2. Определение вариантов использования.....	9
1.3. Вывод.....	10
2. Разработка архитектуры проекта	11
2.1. Обобщенная структура управления приложением	11
2.2. Диаграммы UML, взаимосвязь их компонентов.....	12
2.3. Описание информационных объектов и ограничений целостности.....	13
2.3.1 Таблица ROLES	14
2.3.2 Таблица USERS.....	14
2.3.3 Таблица PERSONS.....	14
2.3.4 Таблица PATIENTS	15
2.3.5 Таблица EMPLOYEES	15
2.3.6 Таблица TALONS	16
2.3.7 Таблица TREATMENTS	16
2.3.8 Таблица ADDRESSES.....	16
2.3.9 Таблица PERSON_ADDRESS	17
2.3.10 Таблица PASSPORTS	17
2.3.11 Таблица DEPARTMENTS.....	17
2.3.12 Таблица DEPARTMENT_EMPLOYEE	18
2.3.13 Таблица POSITIONS	18
2.3.14 Таблица PRICELIST.....	18
2.3.15 Таблица SUPPLIERS.....	19
2.3.16 Таблица PHARMACY	19
2.3.17 Таблица COMMENTS.....	19
2.4. Вывод.....	20
3. Разработка модели базы данных.....	21
3.1. Создание необходимых объектов	21
3.1.1 Табличные пространства	21
3.1.2. Таблицы.....	21
3.1.3. Пакеты и хранимые процедуры	22
3.1.4 Индексы.....	24

3.1.5 Функции	24
3.1.6. Триггеры.....	25
3.2. Описание используемой технологии	25
3.3. Вывод.....	27
4. Установка, настройка и использование Oracle 19с	28
4.1. Создание ролей для разграничения доступа	28
4.2. Описание процедур экспорта и импорта данных.....	31
4.3. Заполнение таблицы 100000 строк	32
4.4. Тестирование производительности базы данных	34
4.5. Вывод.....	35
5. Тестирование, проверка работоспособности и анализ полученных данных.....	36
5.1. Тестирование клиентской части	36
5.2. Тестирование фармацевтической части	37
5.3. Тестирование врачебной части	38
5.4. Тестирование части менеджера	39
5.5. Проверка работоспособности процедур импорта и экспорта	40
5.6. Вывод.....	42
6. Руководство по использованию программного средства	43
6.1. Установка приложения	43
6.2. Панель менеджера	44
6.3. Клиентская часть приложения	45
6.3. Вывод.....	48
Заключение	49
Список использованных литературных источников	50
Приложение А.....	51
Приложение Б	57

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 некоего 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 хранятся 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 необходимости 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.).

Талон Беларусь ▼ Талоны Платные услуги Медучреждения Анализы Лекарства Блог Войти

33-я городская студенческая поликлиника г. Минска
г. Минск, ул. Сурганова, д. 45, корп. 4
показать на карте

Регистратура:
+375 17 378-30-35,
+375 17 379-40-43,
Студенческая деревня: +375 17 377-18-22

**ПН-ПТ: 7:30-20:00
СБ: 9:00-15:00**

33gsp.by
info@33gsp.by

Учреждение может оказывать платные услуги

Заказ талона
Выбор специальности

Нет нужной специальности? Возможно, она есть в платных услугах

Врач общей практики

Врач общей практики («Студенческая деревня», ул. Чюрлениса, 1)

Врач-акушер-гинеколог

Заказ талона

- Специальность
- Врач
- Дата и время
- Личные данные
- Запись

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.

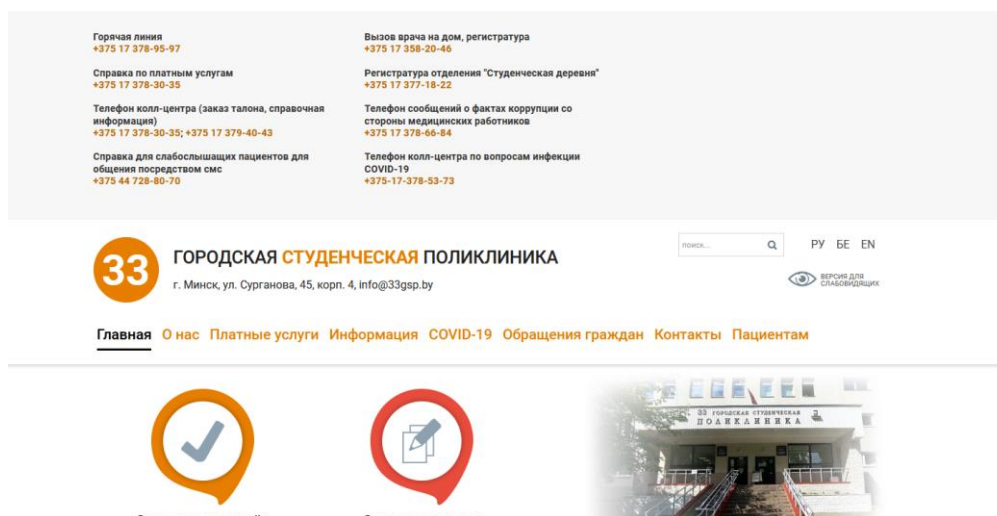


Рисунок 1.2 – Страница сайта студенческой поликлиники

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.

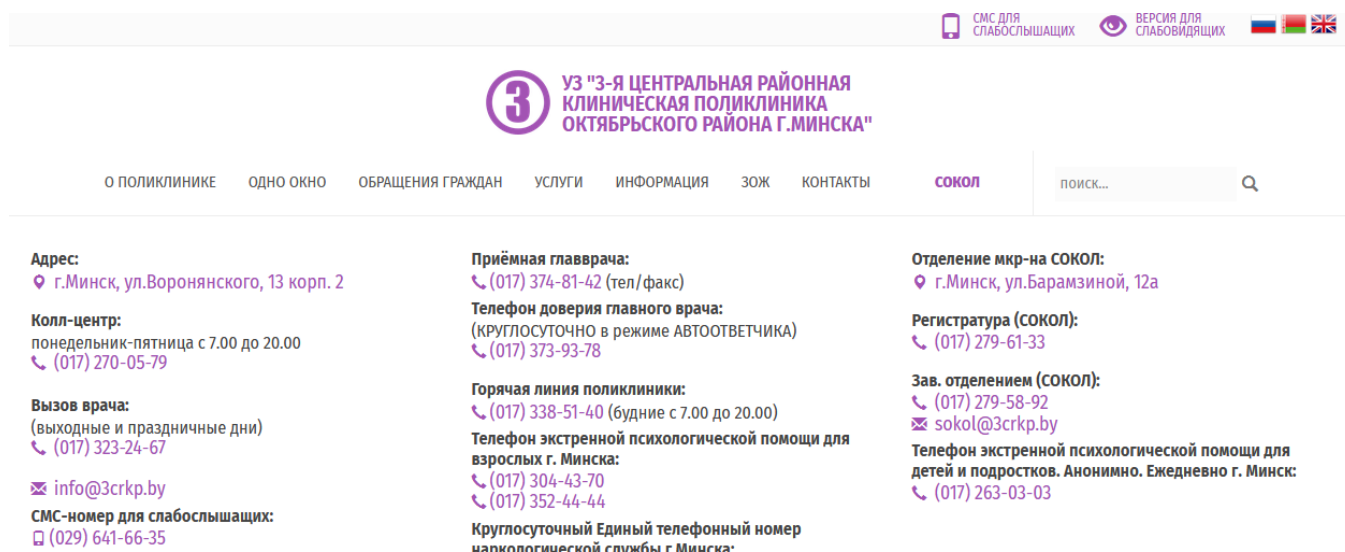


Рисунок 1.3 – Главная страница 3-ей поликлиники

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 расположениях, 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.

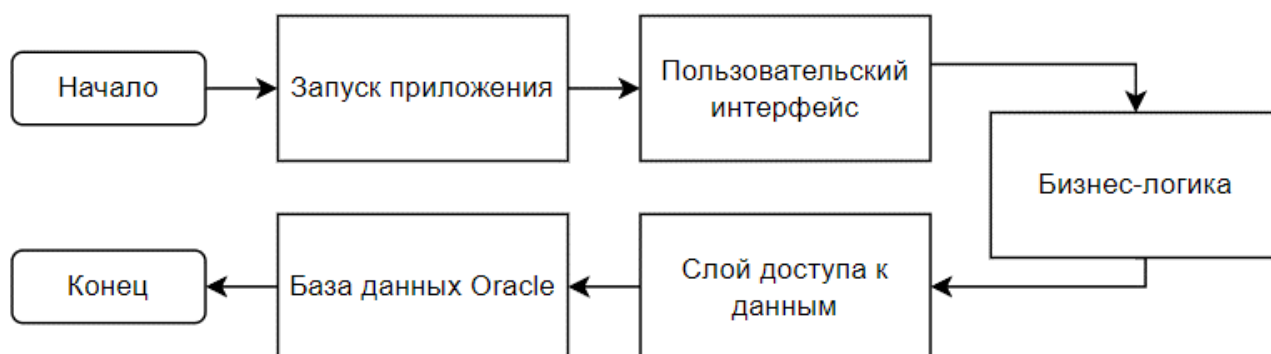


Рисунок 2.1 – Обобщенная структура управления приложением

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 данным ролям добавляются ещё две роли, которые реализует поведение сотрудника медицинского центра – врач и фармацевт.

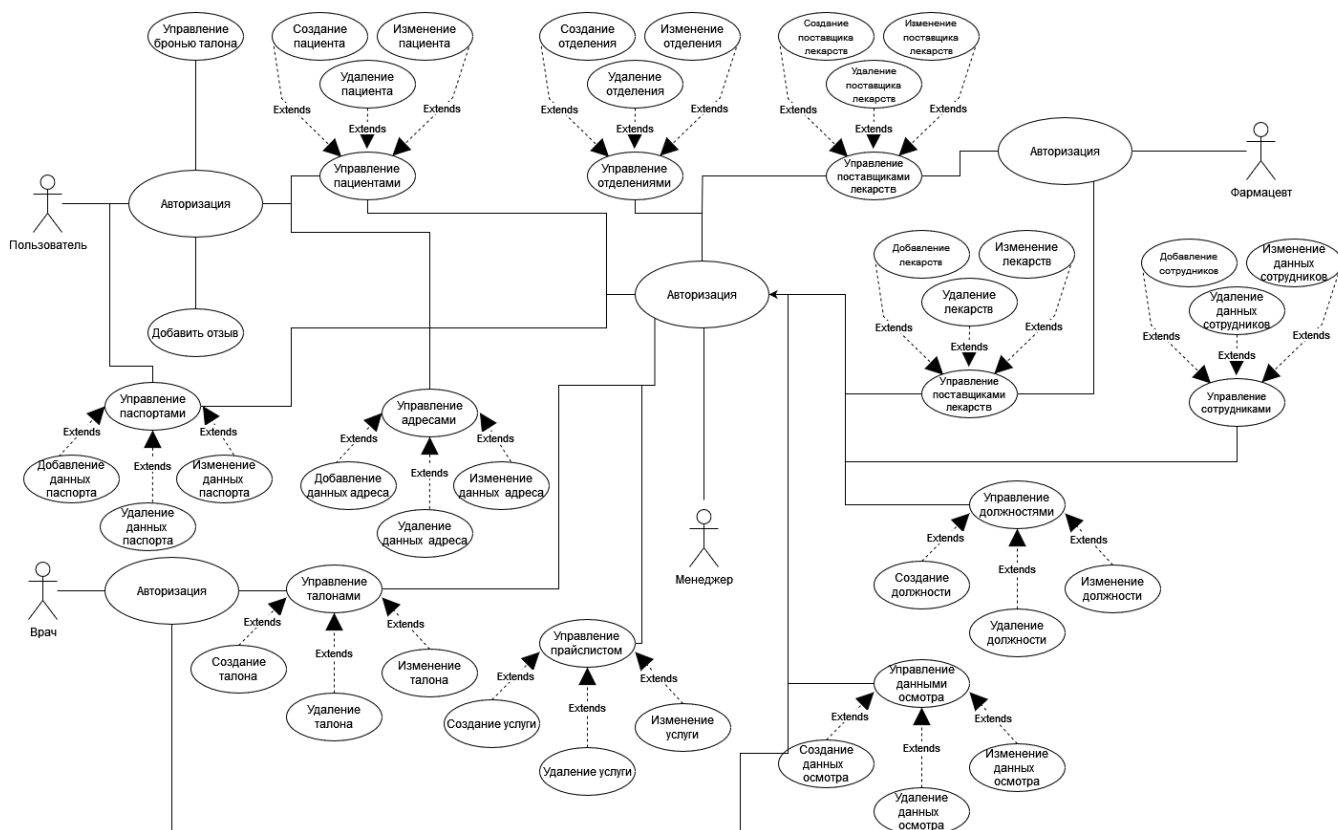


Рисунок 2.2 – Диаграмма вариантов использования

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.

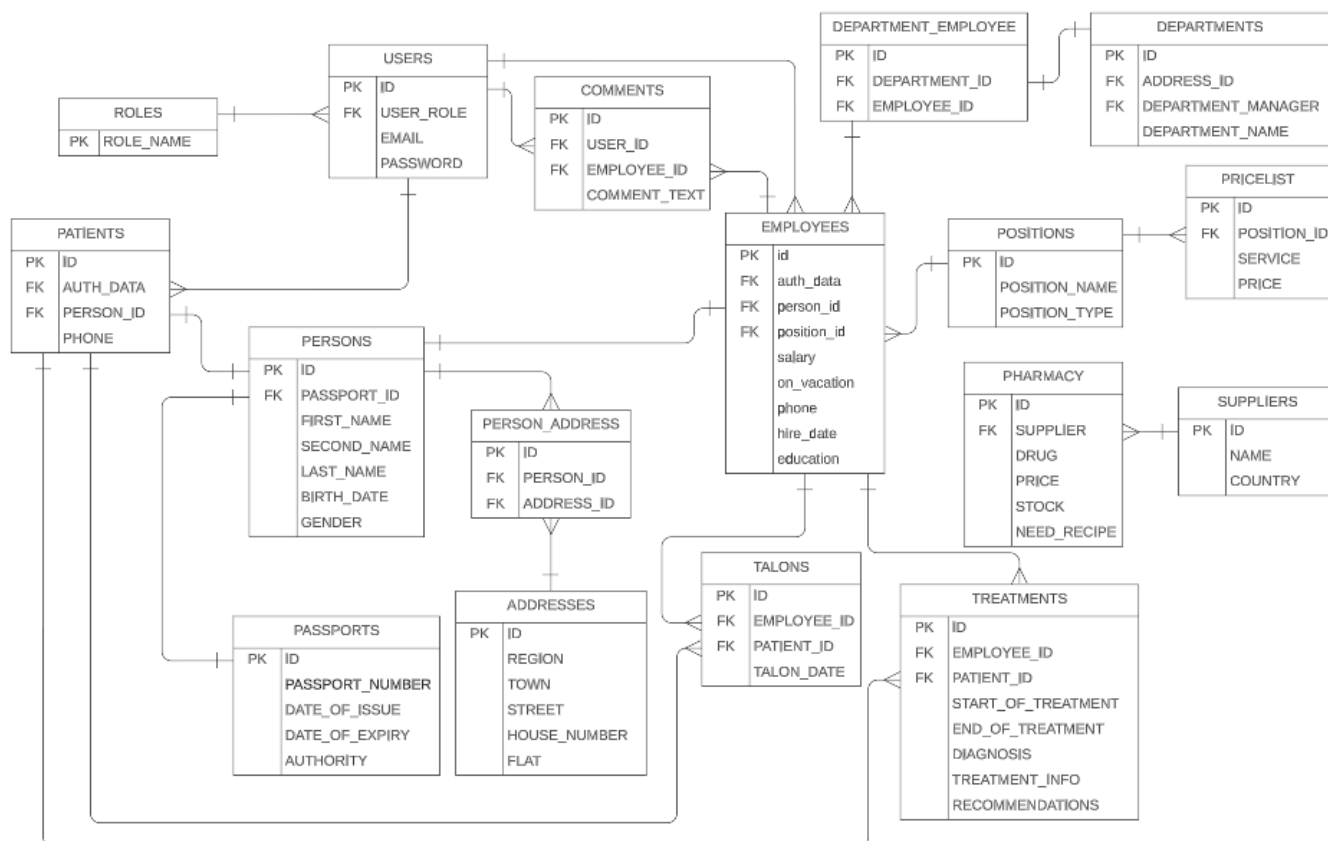


Рисунок 2.3 – Структурная диаграмма базы данных

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.
- DEPARTMENT_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 этапа были спроектированы все необходимые 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 отделения

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_json. 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 помощью 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

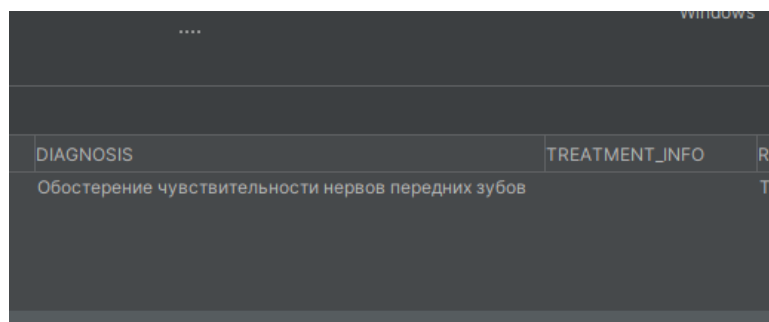
Маскирование предоставляет нам пакет DBMS_REDACT. DBMS_REDACT - это пакет в Oracle Database, который предоставляет интерфейс для маскирования данных, возвращаемых из запросов, выполняемых низкопривилегированными пользователями или приложением. Он позволяет скрыть часть данных от пользователей, которым 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.



DIAGNOSIS	TREATMENT_INFO	RE
Обострение чувствительности нервов передних зубов	*****	Ta

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 посмотрим время выполнения.

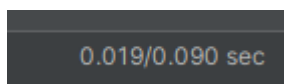


Рисунок 4.1 – Время выполнения запроса без индекса

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

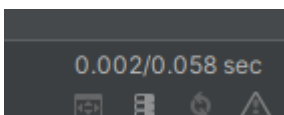


Рисунок 4.2 – Время выполнения запроса с индексом

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.

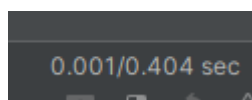


Рисунок 4.3 – Время выполнения нового запроса без индексов

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

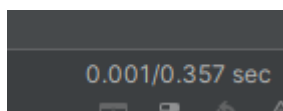


Рисунок 4.4 – Время выполнения нового запроса с индексом на один столбец

Теперь оценим время выполнения запроса с наличием комбинированного индекса.

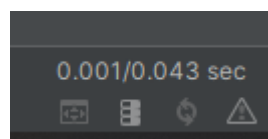


Рисунок 4.5 – Время выполнения нового запроса с комбинированным индексом

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 маленькой 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.

```
SQL> execute SYSTEM.USERS_tapi.create_user('user', 'mail@mon.col', 'user123');
```

Рисунок 5.1 – Создание аккаунта у пользователя

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

ID	NAME	EMAIL	PHONE	ADDRESS	DATE	TIME	STATUS
6	user	mail@mon.col			E606E38B0D8C19B24CF0EE3808183162EA7CD63FF7912DBB22B5E803286B4446		

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.

```
SQL> execute system.book_talon(1, 1);
PL/SQL procedure successfully completed.
```

Рисунок 5.3 – Процедура бронирования талона

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

ID	TALON_DATE	EMPLOYEE_ID	PATIENT_ID
1	2023-05-17 15:25:00	1	1

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.

```
SQL> execute system.suppliers_tapi.create_supplier('Belmed', 'Беларусь');
PL/SQL procedure successfully completed.
```

Рисунок 5.5 – Выполнение процедуры на создание поставщика

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

*	ID	SUPPLIER_NAME	SUPPLIER_COUNTRY
1	100001	Belmed	Беларусь

Рисунок 5.6 – Новая запись в таблице поставщиков

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

```
1 --/
2 BEGIN
3     system.pharmacy_tapi.create_drug('Анальгин', 15.96, 120, '0', 100001);
4 END;
5 --/
```

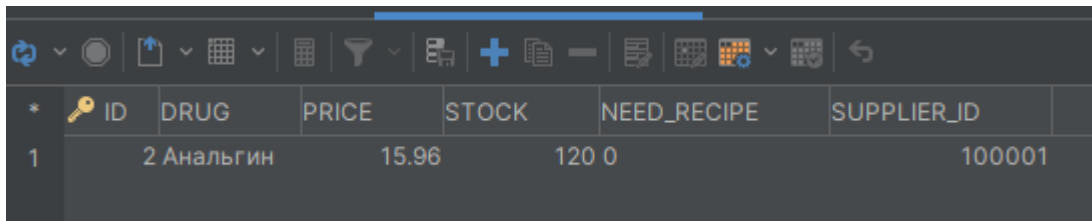
5:4 [95] INS

Log DBMS Output

Time	Status	Command	Exec	Fetch	Rows	Message
23:22:33	STARTED					Executing for: 'Medcenter' [O
23:22:33	SUCCESS	BEGIN	0.014		0	OK. No rows were affected
23:22:33	FINISHED		0.014	0	0	Success: 1

Рисунок 5.7 – Успешное выполнение процедуры

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



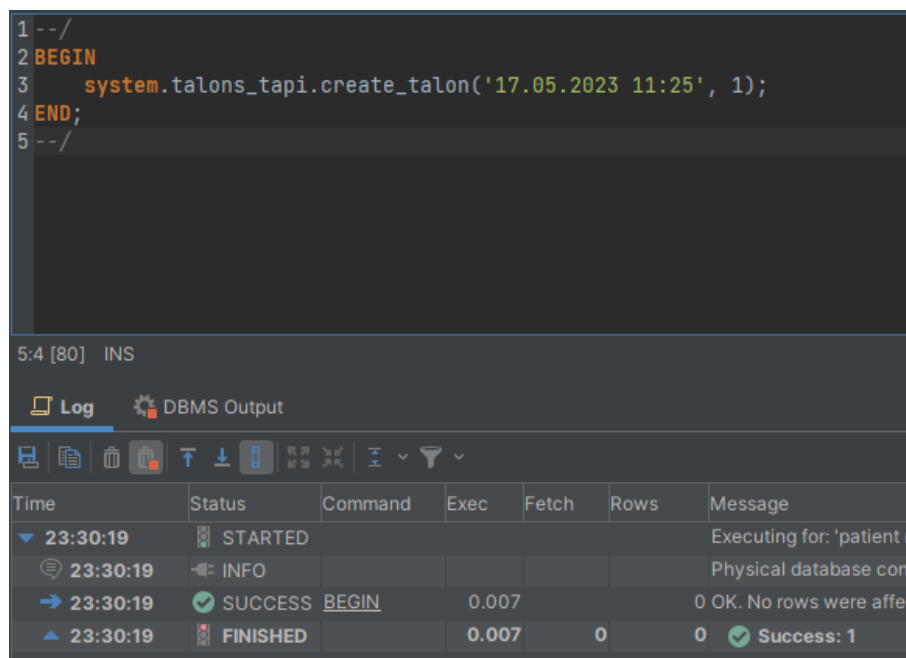
ID	DRUG	PRICE	STOCK	NEED_RECIPE	SUPPLIER_ID
1	2 Анальгин	15.96	120 0		100001

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.



```

1 --/
2 BEGIN
3   system.talons_tapi.create_talon('17.05.2023 11:25', 1);
4 END;
5 --/

```

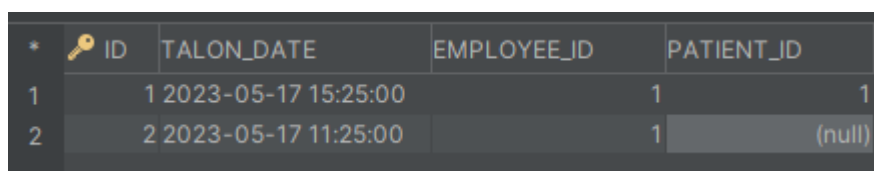
5:4 [80] INS

Log DBMS Output

Time	Status	Command	Exec	Fetch	Rows	Message
23:30:19	STARTED					Executing for: 'patient n
23:30:19	INFO					Physical database con
23:30:19	SUCCESS	BEGIN	0.007			0 OK. No rows were affec
23:30:19	FINISHED		0.007	0	0	Success: 1

Рисунок 5.9 – Успешно выполнение процедуры

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



ID	TALON_DATE	EMPLOYEE_ID	PATIENT_ID
1	2023-05-17 15:25:00	1	1
2	2023-05-17 11:25:00	1	(null)

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.

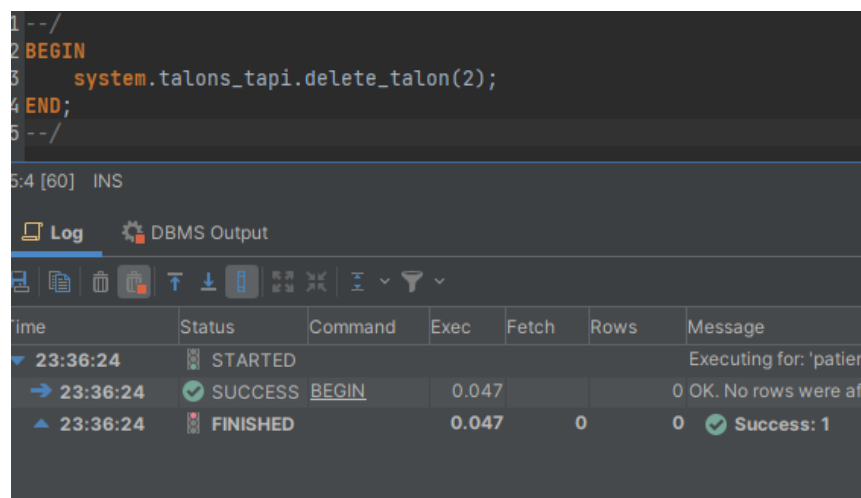


Рисунок 5.11 – Успешное выполнение процедуры

ID	TALON_DATE	EMPLOYEE_ID	PATIENT_ID
1	2023-05-17 15:25:00	1	1

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.

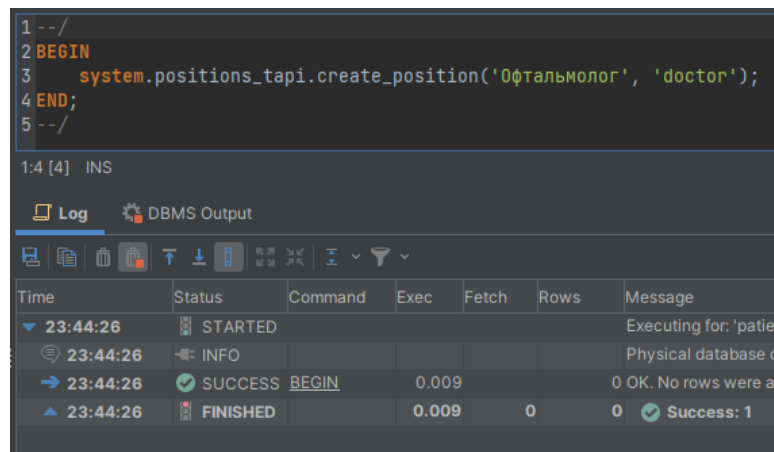


Рисунок 5.13 – Успешное выполнение процедуры

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

ID	POSITION_NAME	POSITION_TYPE
2	Офтальмолог	doctor
1	Стоматолог-хирург	doctor

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.

1 --/
2 BEGIN
3 system.talons_tapi.update_talon(1, '20.05.2023 14:00', null, null);
4 END;
5 --/

5:4 [92] INS

Log

DBMS Output

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

* ID	TALON_DATE	EMPLOYEE_ID	PATIENT_ID
1	1 2023-05-20 14:00:00	1	1

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.

3	STARTED	Executing for: 'Medcenter' [Oracle],
:23	SUCCESS BEGIN	0.007 0 OK. No rows were affected
:23	FINISHED	0.007 0 0 Success: 1

Рисунок 5.17 – Успешное выполнение процедуры экспорта

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

```

{} comments.json X
D: > db > {} comments.json > ...
1  [
2    {
3      "user_id": 4,
4      "comment_text": "1",
5      "employee_id": "1"
6    },
7    {
8      "user_id": 5,
9      "comment_text": "doctor",
10     "employee_id": "1"
11   },
12   {
13     "user_id": 4,
14     "comment_text": "1",
15     "employee_id": "1"
16   },
17   {
18     "user_id": 5,
19     "comment_text": "doctor",
20     "employee_id": "1"
21   }
22 ]

```

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.

```

[
  {
    "user_id": 4,
    "comment_text": "Изменённое значение для проверки",
    "employee_id": "1"
  },
]

```

Рисунок 5.19 – Содержимое json-файла

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


 ID	USER_ID	EMPLOYEE_ID	COMMENT_TEXT
7	4	1	1
8	5	1	doctor
1	4	1	1
2	5	1	doctor
9	4	1	Изменённое значение для проверки
10	5	1	doctor
11	4	1	1
12	5	1	doctor

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.

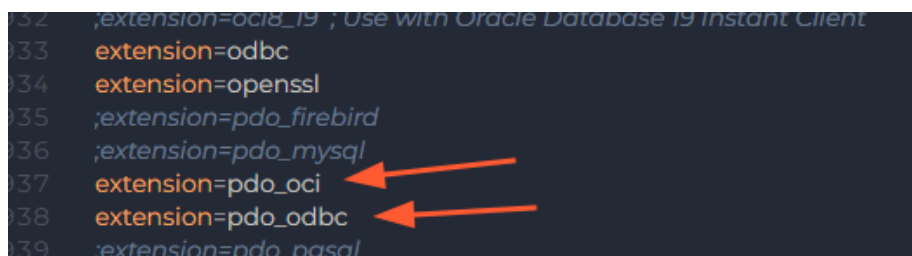
Если говорить в общем, то можно сделать следующий вывод: наличие отлично построенных объектов базы данных (процедур, функций, индексов и т.д.) показывает отличный уровень эффективности в работе с самой базой данных.

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.



```

32 ;extension=oci8_19 ; Use with Oracle Database 19 Instant Client
33 extension=odbc
34 extension=openssl
35 ;extension=pdo_firebird
36 ;extension=pdo_mysql
37 extension=pdo_oci
38 extension=pdo_odbc
39 ;extension=pdo_pgsql
  
```

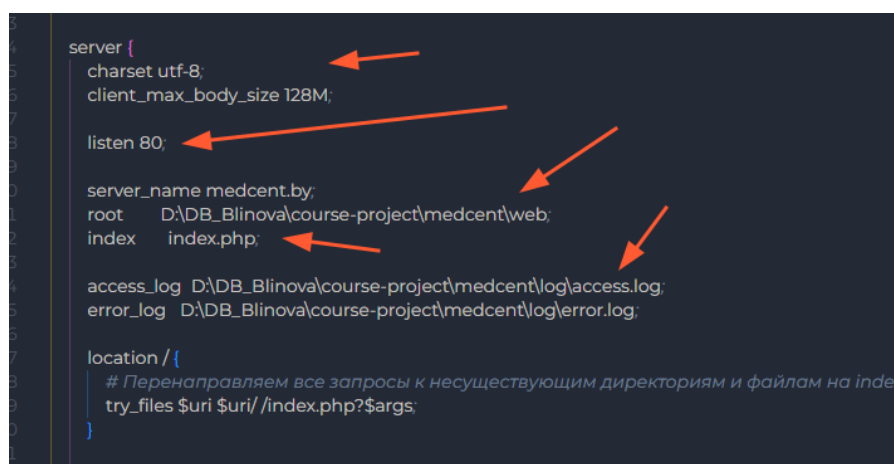
Рисунок 6.1 – Файл конфигурации PHP

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.



```

server {
    charset utf-8;
    client_max_body_size 128M;

    listen 80;

    server_name medcent.by;
    root D:\DB_Blinova\course-project\medcent\web;
    index index.php;

    access_log D:\DB_Blinova\course-project\medcent\log\access.log;
    error_log D:\DB_Blinova\course-project\medcent\log\error.log;

    location / {
        # Перенаправляем все запросы к несуществующим директориям и файлам на index.php
        try_files $uri $uri/ /index.php?$args;
    }
}
  
```

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 «Congratulations», значит фреймворк установлен правильно.

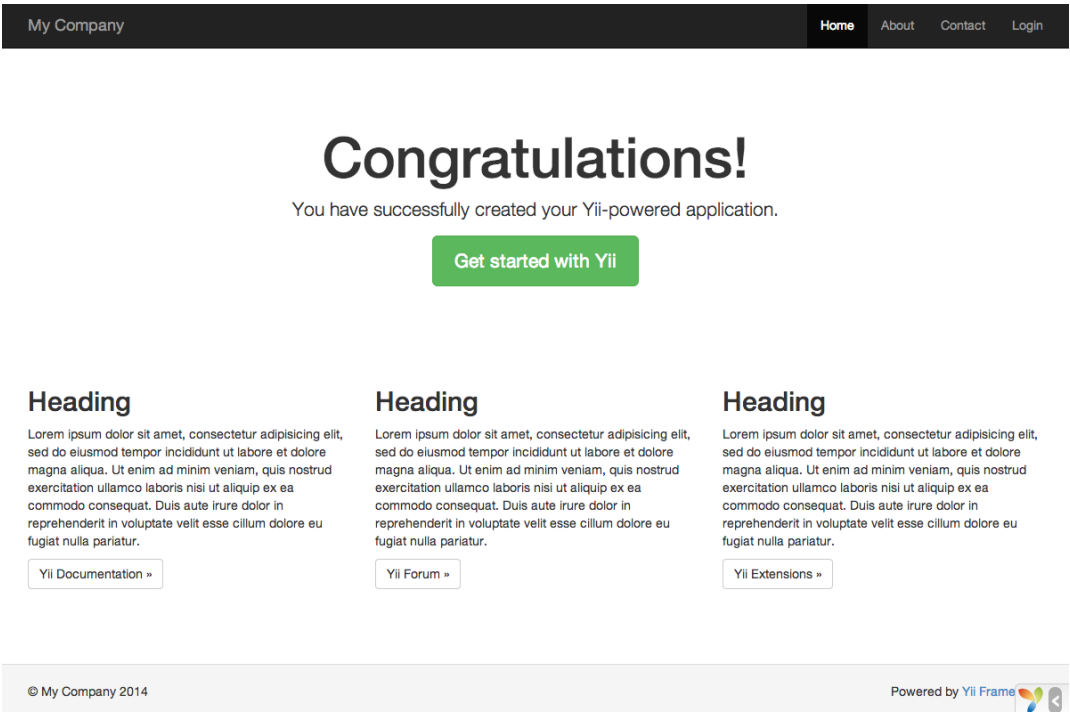


Рисунок 6.3 – Главная страница фреймворка Yii2

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.

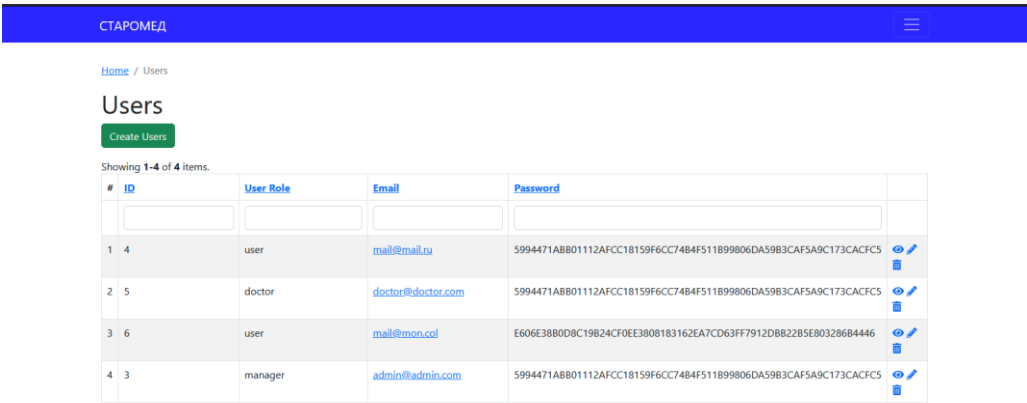


Рисунок 6.4 – Панель менеджера

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

СТАРОМЕД			
Users	Addresses	Passports	Persons
Patients	Roles	Employees	Departments
DepartmentEmployee	Suppliers	Pharmacy	Talons
Treatments	Pricelist	Person Address	Positions
Comments			

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.

СТАРОМЕД

Заказать талон Прайслист Аптека Оставить отзыв Войти

[Home](#) / Регистрация

Регистрация

Please fill out the following fields to signup:

Email

Пароль

Создать аккаунт

Уже есть аккаунт? Войти

Рисунок 6.6 – Страница регистрации

На следующем изображении представлена страница авторизации пользователей.

СТАРОМЕД

Заказать талон Прайслист Аптека Оставить отзыв Войти

[Home](#) / Авторизация

Авторизация

Email

Вы кое-что забыли!

Пароль

☒ Запомнить меня

Войти

Зарегистрироваться

Рисунок 6.7 – Страница авторизации

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

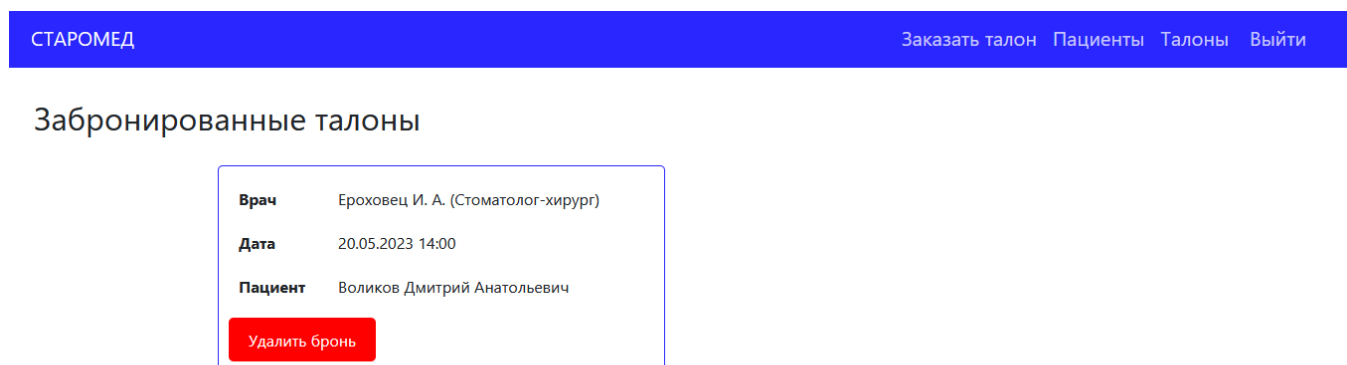


Рисунок 6.8 – Главная страница аккаунта пользователя

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.

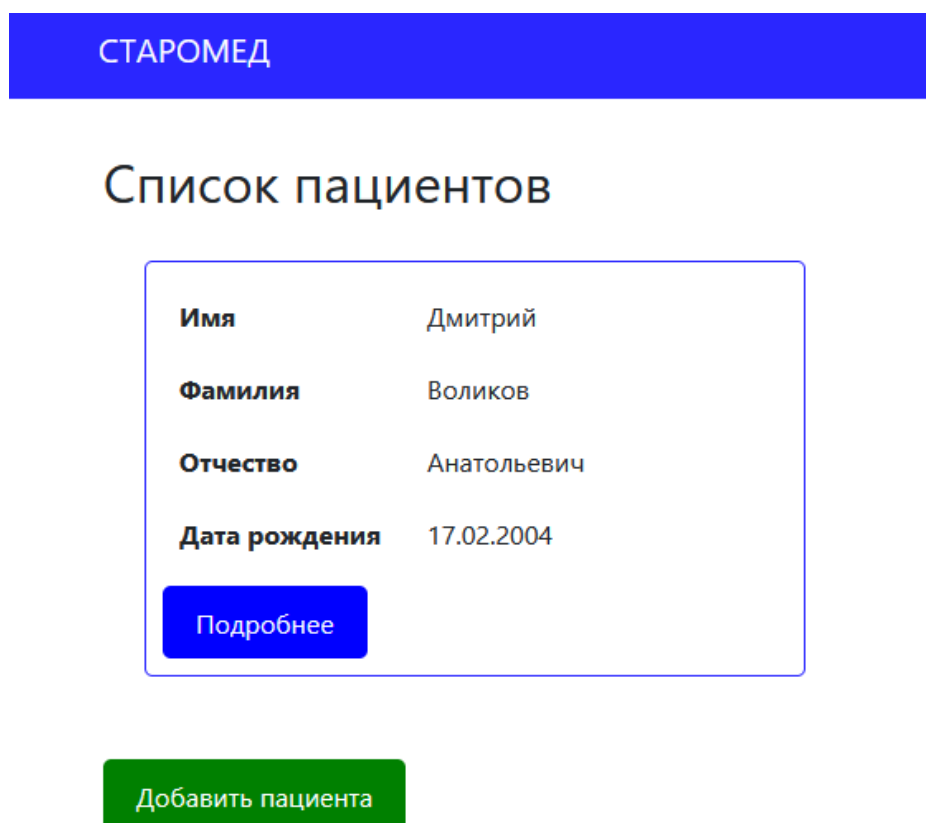


Рисунок 6.9 – Страница пациентов пользователя

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

СТАРОМЕД

Заказать талон Пациенты Талоны Выйти

First Name

Дмитрий

Second Name

Воликов

Last Name

Анатольевич

Birth Date

17.02.2004

Gender

Male

Phone

80 (29) 254-36-89

Сохранить

Рисунок 6.10 – Страница изменения персональных данных пациента

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

Область

Выберите область

Населённый пункт

Улица

Номер дома

Квартира (если есть)

Save

Рисунок 6.11 – Форма добавления адреса к пациенту

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

СТАРОМЕД

Заказать талон Прайслист Аптека Оставить отзыв Аккаунт

Заказать талон

Стоматолог-хирург (обл. Минск, Минск, ул. Чюрлёниса, д. 1)

Врач

Докторов И. А.

Дата

20.05.2023 14:00

Заказать

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 надёжной 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 данных 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 операций 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. Официальный сайт Oracle [Электронный ресурс] / Справочник Режим доступа: <https://www.oracle.com/> – Дата доступа 6.04.2023.
7. Официальная 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 [Электронный 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;

```

```

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