

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
"ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ"

Кафедра программной инженерии

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ

ПО ДИСЦИПЛИНЕ: «Программирование систем с серверами баз данных»
НА ТЕМУ: «Создание клиент-серверной информационной системы
средствами СУБД»

Руководители:

Щедрин С. В.

Ногтев Е. А.

Выполнил:

ст. гр. ПИ-196

Носаченко А. А.



РЕФЕРАТ

Отчет по курсовой работе содержит: 65 страниц, 51 рисунок, 0 таблиц, 5 приложений, 6 источников.

Объект исследования – клиент-серверная информационная система, созданная средствами системы управления базами данных (далее – СУБД), взаимодействующая с реляционной базой данных (далее – БД) по принципу разделения ролей.

Цель – изучение на практике способов проектирования и реализации реляционных баз данных с применением средств СУБД PostgreSQL, ориентированных на одновременное применение несколькими пользователями, имеющими различные роли, а также разработка соответствующей клиент-серверной программной системы для обеспечения взаимодействия пользователей с БД.

Результат – БД, соответствующая теме задания – «Парикмахерские», серверное приложение управления СУБД, клиентское приложение для взаимодействия с сервером, руководства пользователей.

СОДЕРЖАНИЕ

РЕФЕРАТ	2
СОДЕРЖАНИЕ	3
ВВЕДЕНИЕ	5
1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ, ПОСТАНОВКА ЗАДАЧИ	6
2 ОБОСНОВАНИЕ ВЫБОРА СУБД, ОПИСАНИЕ ВОЗМОЖНОСТЕЙ СУБД	8
3 ОБОСНОВАНИЕ ВЫБОРА ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ ДЛЯ НАПИСАНИЯ КЛИЕНТСКОЙ ЧАСТИ, ПРОЕКТИРОВАНИЕ СТРУКТУРЫ ПО..	11
3.1 Невизуальные компоненты для работы с данными.....	11
3.2 Визуальные компоненты отображения данных.....	12
3.3 Разработка шаблонов приложений для работы с таблицами базы данных...	14
4 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ В ВЫБРАННОЙ СУБД.....	15
4.1 Проектирование концептуальной модели БД.....	15
4.2 Создание таблиц, доменов, индексов, сиквенсов	16
4.3 Разработка триггеров	17
4.4 Организация многоролевого доступа к данным	19
4.5 Разграничение доступа к данным на уровне строк	20
4.6 Партицирование одной из основных таблиц БД	21
4.7 Проектирование запросов к базе данных.....	22
4.8 Создание представлений и хранимых процедур, функций.....	28
5 РАЗРАБОТКА КЛИЕНТСКОГО ПРИЛОЖЕНИЯ	32
5.1 Формы и компоненты для работы в роли «Менеджер»	33
5.2 Формы и компоненты для работы в роли «Директор».....	34
5.3 Формы и компоненты для работы в роли «Работник»	35
5.4 Генерация результатов не менее трех итоговых запросов (диаграммы, экспорт в Excell)	36
6 ТЕСТИРОВАНИЕ РАЗРАБОТАННОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ.....	38
ВЫВОДЫ	39
СПИСОК ЛИТЕРАТУРЫ.....	40

ПРИЛОЖЕНИЕ А. Техническое задание	41
ПРИЛОЖЕНИЕ Б. Листинг серверного приложения	47
ПРИЛОЖЕНИЕ В. Листинг клиентского приложения	54
ПРИЛОЖЕНИЕ Г. Руководство пользователя.....	64
ПРИЛОЖЕНИЕ Д. Руководство администратора	65

ВВЕДЕНИЕ

В наше время сложно найти фирму, которая бы не использовала персональные компьютеры для хранения и обработки информации. Одним из лучших способов хранения различной информации является внесение её в базу данных. Всё, с чем мы взаимодействуем в жизни, вероятно, зафиксировано в какой-нибудь базе. Базы данных формируются и работают под управлением специальных программных средств, называемых системами управления базами данных.

База данных – это организованная структура, которая предназначена для хранения информации. В то время, когда происходило развитие термина баз данных, в них сохранялись исключительно информация, однако сейчас многие системы управления базами данных позволяют размещать в своих структурах и данные, и программный код, с помощью которого совершается связь с пользователями или с другими программно-аппаратными комплексами [1]. При этом данные должны не противоречить друг другу, быть целостными и не избыточными. База данных создается для сохранения и непосредственного доступа к информации, содержащей сведения об искомой предметной области.

Система управления базами данных – это программный механизм, предназначенный для записи, поиска, сортировки, обработки и печати информации, содержащейся в базе данных [2].

Накопление хранимого объема информации, рост группы пользователей информационных систем служат источником к обширному развитию комфортных в интерфейсе и относительно лёгких для понимания табличных систем управления базами данных. Создание доступа к информации базы данных сразу нескольких пользователей одновременно, зачастую находящихся на далеком расстоянии от места хранения баз данных, а также друг от друга, в настоящее время является наиболее актуальной проблемой разрабатываемых систем, использующих базы данных. В них решаются проблемы характерные для параллельных процессов, правильности данных, а также получения не санкционированного входа.

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ, ПОСТАНОВКА ЗАДАЧИ

В ходе курсовой работы необходимо разработать базу данных с приложения взаимодействия с ней для учета работы парикмахерских города, которая будет содержать информацию о парикмахерских (номер, район города, разряд (высший, первый, второй), тип собственности (частная, государственная, акционерная,...), год начала функционирования, телефон), клиентах (ФИО, дата рождения, социальная группа (предприниматель, банковский служащий, инженер, рабочий,...), домашний адрес)) и оказанных клиентам услугах (парикмахерская, название услуги (стрижка, завивка, укладка, маникюр, массаж,...), стоимость (зависит от разряда парикмахерской), дата оказания услуги).

Исходя из данной задачи, необходимо выполнить следующие этапы разработки:

1. Спроектировать концептуальную модель базы данных (БД) для предметной области и представить ее в виде взаимосвязанных таблиц, находящихся в третьей нормальной форме. Выделить базовые таблицы и таблицы-справочники, указать для них первичные и внешние ключи.

2. Создать базу данных в среде СУБД средствами языка SQL. Добавить таблицы, домены, индексы.

3. Разработать не менее шести триггеров (по одному для каждого типа события), как минимум для двух различных таблиц БД. Триггеры типа BEFORE INSERT должны быть созданы для всех таблиц и с использованием генераторов задавать значение первичного ключа для вновь добавляемой записи.

4. Заполнить таблицы БД с использованием соответствующих запросов на языке SQL.

5. Сформулировать следующие виды запросов:

- симметричное внутреннее соединение с условием (два запроса с условием отбора по внешнему ключу, два – по датам);
- симметричное внутреннее соединение без условия (три запроса);
- левое внешнее соединение;
- правое внешнее соединение;
- запрос на запросе по принципу левого соединения;

- итоговый запрос без условия;
- итоговый запрос без условия с итоговыми данными вида: «всего», «в том числе»;
- итоговые запросы с условием на данные (по значению, по маске, с использованием индекса, без использования индекса);
- итоговый запрос с условием на группы;
- итоговый запрос с условием на данные и на группы;
- запрос на запросе по принципу итогового запроса;
- запрос с использованием объединения
- запросы с подзапросами.

6. Запросы без параметров реализовать в виде представлений, остальные запросы – в виде хранимых процедур и/или функций. Создать, по меньшей мере, одно модифицируемое представление, используя механизм триггеров. ВСЯ логика проектируемого ПО – на сервере.

7. Разработать клиентское приложение, которое предоставляет следующие возможности для работы с созданной базой данных:

- многопользовательский режим работы (одна программа для всех ролей – ситуативный доступ к интерфейсу)
- наличие нескольких ролей пользователя (менеджер – добавление/удаление/редактирование пользователей, их прав/ролей; директор – просмотр отчётов о прибыли и убытках, работник – создание записей о проводимых работах, изучение личного дохода)
- просмотр содержимого таблиц и представлений (здесь и далее – с учетом прав пользователей);
- добавление, редактирование и удаление записей таблиц и модифицируемых представлений;
- работа с наборами данных, находящимися в отношении «один-ко-многим» (создать составную форму для просмотра и редактирования данных родительской и дочерней таблиц);
- поиск и фильтрация данных отображаемых таблиц;
- просмотр результатов выполнения запросов;
- визуализация результатов одного из итоговых запросов.

8. Обеспечить защиту данных, информации от несанкционированного доступа, сделать защиту на уровне строк, выполнить партиционирование одной из основных таблиц.

2 ОБОСНОВАНИЕ ВЫБОРА СУБД, ОПИСАНИЕ ВОЗМОЖНОСТЕЙ СУБД

Для разработки системы выбрана СУБД PostgreSQL. Она является одним из нескольких бесплатных популярных вариантов СУБД. Это весьма старая система, поэтому в настоящее время она хорошо развита, и позволяет пользователям управлять как структурированными, так и неструктурированными данными. Может быть использована на большинстве основных платформ, включая Linux (где особенно хорошо проявляется производительность). Прекрасно справляется с задачами импорта информации из других типов баз данных с помощью собственного инструментария. Движок БД может быть размещен в ряде сред, в том числе виртуальных, физических и облачных.

Достоинства СУБД:

- является масштабируемым решением и позволяет обрабатывать терабайты данных;
- поддерживает формат json;
- существует множество predefined функций;
- доступен ряд интерфейсов;
- поддержка БД неограниченного размера;
- мощные и надёжные механизмы транзакций и репликации;
- расширяемая система встроенных языков программирования и поддержка загрузки C-совместимых модулей;
- наследование;
- легкая расширяемость.

Идеально подходит для организаций с ограниченным бюджетом, требует привлечения квалифицированных специалистов, когда требуется возможность выбрать уникальный интерфейс и использовать json.

Кроме основных возможностей, присущих любой SQL базе данных, PostgreSQL поддерживает:

- Очень высокий уровень соответствия ANSI SQL 92, ANSI SQL 99 и ANSI SQL 2003.

- Схемы, которые обеспечивают пространство имен на уровне SQL. Схемы содержат таблицы, в них можно определять типы данных, функции и операторы.

- Subqueries – подзапросы (subselects), полная поддержка SQL92. Подзапросы делают язык SQL более гибким и зачастую более эффективным.

- Outer Joins – внешние связки (LEFT, RIGHT, FULL)

- Rules – правила, согласно которым модифицируется исходный запрос.

- Views – представления, виртуальные таблицы. Реальных экземпляров этих таблиц не существуют, они материализуются только при запросе.

- Cursors – курсоры, позволяют уменьшить трафик между клиентом и сервером, а также память на клиенте, если требуется получить не весь результат запроса, а только его часть.

- Table Inheritance – наследование таблиц, позволяющее создавать объекты, которые наследуют структуру родительского объекта и добавлять свои специфические атрибуты.

- Prepared Statements (преподготовленные запросы) – это объекты, живущие на стороне сервера, которые представляют собой оригинальный запрос после команды PREPARE, который уже прошел стадии разбора запроса (parser), модификации запроса (rewriting rules) и создания плана выполнения запроса (planner), в результате чего, можно использовать команду EXECUTE, которая уже не требует прохождения этих стадий. Для сложных запросов это может быть большим выигрышем.

- Stored Procedures – серверные (хранимые) процедуры позволяют реализовывать бизнес логику приложения на стороне сервера. Кроме того, они позволяют сильно уменьшить трафик между клиентом и сервером.

- Savepoints(nested transactions) – в отличие от "плоских транзакций", которые не имеют промежуточных точек фиксации, использование savepoints позволяет отменять работу части транзакции, например в следствии ошибочно введенной команды, без влияния на оставшуюся часть транзакции.

- Права доступа к объектам системы на основе системы привилегий.

- Система обмена сообщениями между процессами – LISTEN и NOTIFY позволяют создать событийную модель взаимодействия между клиентом и сервером.

- Триггеры позволяют управлять реакцией системы на изменение данных, как перед самой операцией (BEFORE), так и после (AFTER).

- Cluster table – упорядочивание записей таблицы на диске согласно индексу, что иногда за счет уменьшения доступа к диску ускоряет выполнение запроса.

Богатый набор типов данных PostgreSQL включает:

- Символьные типы (character(n)) как определено в стандарте SQL и тип text с практически неограниченной длиной.

- Numeric тип поддерживает произвольную точность.

- Массивы согласно стандарту SQL:2003.

- Большие объекты (Large Objects) для бинарных данных размером до 2Gb.

- Геометрические типы для работы с пространственными данными на плоскости.

- ГИС (GIS) типы в PostgreSQL позволяют работать с трехмерными данными.

- Сетевые типы (Network types).

- Композитные типы (composite types).

- Типы времени реализованы с очень большой точностью.

- Псевдотипы serial и bigserial организующие AUTO_INCREMENT.

Безопасность данных также является важнейшим аспектом любой СУБД. В PostgreSQL она обеспечивается 4-мя уровнями безопасности:

- PostgreSQL нельзя запустить под привилегированным пользователем – системный контекст

- SSL,SSH шифрование трафика между клиентом и сервером – сетевой контекст

- Сложная система аутентификации на уровне хоста или IP адреса/подсети.

Система аутентификации поддерживает пароли, зашифрованные пароли, Kerberos, IDENT и прочие системы, которые могут подключаться, используя механизм подключаемых аутентификационных модулей.

- Детализированная система прав доступа ко всем объектам базы данных, которая совместно со схемой, обеспечивающая изоляцию названий объектов для каждого пользователя, PostgreSQL предоставляет богатую и гибкую инфраструктуру.

3 ОБОСНОВАНИЕ ВЫБОРА ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ ДЛЯ НАПИСАНИЯ КЛИЕНТСКОЙ ЧАСТИ, ПРОЕКТИРОВАНИЕ СТРУКТУРЫ ПО

3.1 Невизуальные компоненты для работы с данными

Для чтения данных из базы была применена библиотека Npgsql, которая обеспечивает взаимодействие баз данных, основанных на СУБД Postgres, и приложений на языке C#. На рисунке 3.1 приведен пример кода чтения справочной таблицы.

```
case "C24": // SELECT Groups
{
    try
    {
        var list = db.Two_List.FromSqlRaw($"SELECT * FROM C24_2_fields()").ToList();

        StringBuilder builder = new StringBuilder();
        foreach (var item in list)
        {
            if (item.Field_1 != "0") builder.Append($"{r\n~{item.Field_1}~{item.Field_2}");
        }
        message = builder.ToString();
    }
    catch (Exception ex)
    {
        message = $"{ex.Message} . EXCEPTION";
        Console.WriteLine($"{db.User_name}: {message}");
    }
    Send(message);
}
break;
```

Рисунок 3.1 – Код чтения таблицы «Группы»

Библиотека Npgsql позволяет создавать клиент-серверное соединение приложения с базой данных, а также поддерживает контроль базы данных под управлением пользователя базы, под которым произведено подключение, в форме выполнения запросов, а также прямое обращение к объектам и таблицам базы данных посредством возможности отражения их в виде объектов приложения (см. рис. 3.2).

```

ссылка: 3 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
public class DB_Process : DbContext
{
    ссылка: 2 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
    public class ToOne
    {
        ссылка: 14 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
        public string Field_1 { get; set; }
    }
    ссылка: 2 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
    public class ToTwo
    {
        ссылка: 2 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
        public class ToThree
        {
            ссылка: 2 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
            public class ToFour
            {
                ссылка: 2 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
                public class ToFive
                {
                    ссылка: 2 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
                    public class ToSix
                    {
                        ссылка: 2 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
                        public class ToSeven
                        {
                            ссылка: 2 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
                            public class ToEight
                            {
                                ссылка: 49 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
                                public string User_name { get; set; }
                                ссылка: 2 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
                                public string Password { get; set; }

                                ссылка: 14 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
                                public DbSet<ToOne> One_List { get; set; }
                                ссылка: 14 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
                                public DbSet<ToTwo> Two_List { get; set; }
                                ссылка: 7 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
                                public DbSet<ToThree> Three_List { get; set; }
                                ссылка: 2 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
                                public DbSet<ToFour> Four_List { get; set; }
                                ссылка: 0 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
                                public DbSet<ToFive> Five_List { get; set; }
                                ссылка: 0 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
                                public DbSet<ToSix> Six_List { get; set; }
                                ссылка: 2 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
                                public DbSet<ToSeven> Seven_List { get; set; }
                                ссылка: 1 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
                                public DbSet<ToEight> Eight_List { get; set; }

                                ссылка: 1 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
                                public DB_Process(string name, string password)
                                {
                                    ссылка: 0 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
                                    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
                                    {
                                        optionsBuilder.UseNpgsql($"Host=localhost;Port=5432;Database=PSSSBD;Pooling=false;Username={User_name};Password={Password}");
                                    }

                                    ссылка: 0 | Keronon S, 16 ч назад | Автор: 1, изменение: 1
                                    protected override void OnModelCreating(ModelBuilder modelBuilder)
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

Рисунок 3.2 – Класс-обработчик базы данных

3.2 Визуальные компоненты отображения данных

Для визуального отображения данных и диалога с пользователем применяется встроенная библиотека WinForms, обеспечивающая следующие объекты пользовательского интерфейса:

DataGridView – область отображения данных в виде таблицы, используется клиентским приложением для вывода получаемой из базы данных информации, поддерживает множественный выбор строк и закрепление за элементом меню быстрого доступа;

TextBox – текстовое поле с возможностью ввода данных, активно применяется в приложении для диалога с пользователем, обеспечивая ввод необходимых данных, при меняемых при вызове различных функций;

ComboBox – выпадающий список, позволяющий пользователю выбрать одно из предустановленных значений, используется в формах ввода и изменения записей в базе данных, позволяет пользователю сделать выбор из связанных внешними ключами таблиц, не опасаясь сделать ошибку и получить исключение нарушения ключа (см. рис. 3.3);

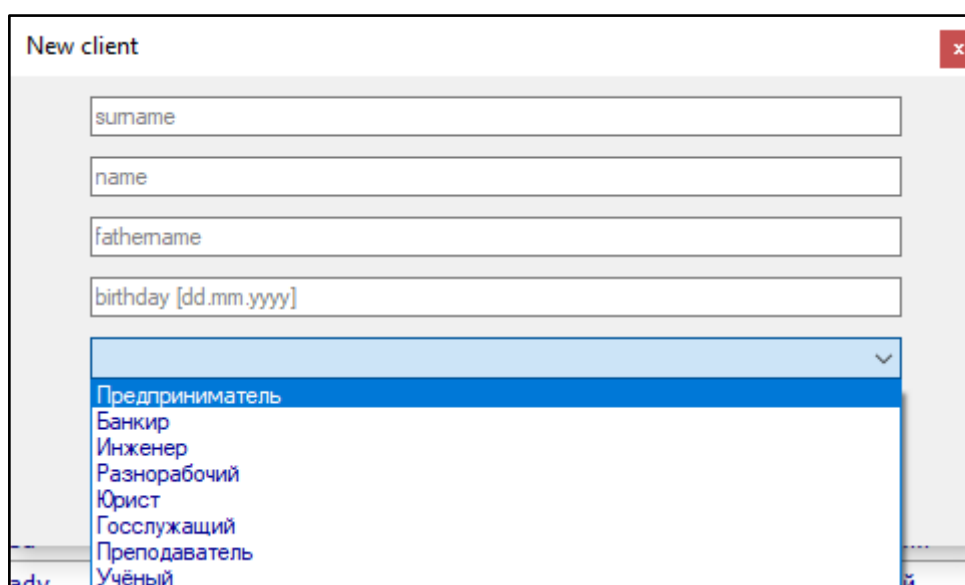
The image shows a screenshot of a software window titled "New client". The window has a standard Windows-style title bar with a close button (red 'X') in the top right corner. Inside the window, there are four text input fields stacked vertically, labeled "surname", "name", "fathename", and "birthday [dd.mm.yyyy]". Below these fields is a dropdown menu (ComboBox) with a blue header bar and a downward arrow on the right. The dropdown is open, showing a list of professions in Russian: "Предприниматель", "Банкир", "Инженер", "Разнорабочий", "Юрист", "Госслужащий", "Преподаватель", and "Учёный". The "Предприниматель" option is currently selected and highlighted in blue.

Рисунок 3.3 – Пример применения текстовых полей и выпадающих списков

MessageBox – объект, представляющий из себя простое окно приложения, состоящее из заголовка окна, а также теста и кнопок подтверждения выбора в теле окна, используется для оповещения пользователя о событиях системы, а также предоставляет выбор при активации функции удаления записей;

NumericUpDown – текстовое поле с парой небольших кнопок, способное хранить только численные значения, применяется для отображения текущей страницы записей в поле-таблице.

3.3 Разработка шаблонов приложений для работы с таблицами базы данных

Следую основам клиент-серверного разделения функционала системы и исходя из задачи по обеспечению достаточной безопасности системы, можно выделить следующие концепции проектируемого программного обеспечения:

- У серверной части нет необходимости в интерактивном диалоге с пользователем, потому её проект стоит выполнить в форме консольного приложения;
- Клиентское приложение следует создавать достаточно понятным и удобным для работы с ним пользователя, потому его следует выполнить в виде оконного интерактивного приложения с пользовательским интерфейсом;
- Следует ограничить количество одновременной информации на экране пользователя, для уменьшения нагрузки на него;
- Перед отображением пользователю, информацию из БД следует подготовить должным образом, исключив ненужные данные;
- Следует ограничить доступ пользователей к данным, которые находятся вне их компетенций, прав и обязанностей;
- Действия пользователя необходимо отслеживать и проверять на корректность на каждом шаге выполнения.

При разработке форм клиентского приложения планируется делать отдельные логические области интерфейса в виде различных форм, вызываемых при определённых условиях в работе системы, оставляя постоянной только область отображения данных из базы.

4 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ В ВЫБРАННОЙ СУБД

4.1 Проектирование концептуальной модели БД

При проектировании концептуальной модели БД основной задачей становится обеспечение корректной взаимосвязи парикмахерских, множества клиентов, посещающих и множества услуг, которые они могут заказать за одно посещение.

Для выполнения данной задачи следует ввести дополнительный объект БД – таблицу, отражающую журнал посещений.

Исходя из этого схема БД, при приведении её к нормальной форме, примет вид, отражённый на рисунке 4.1.

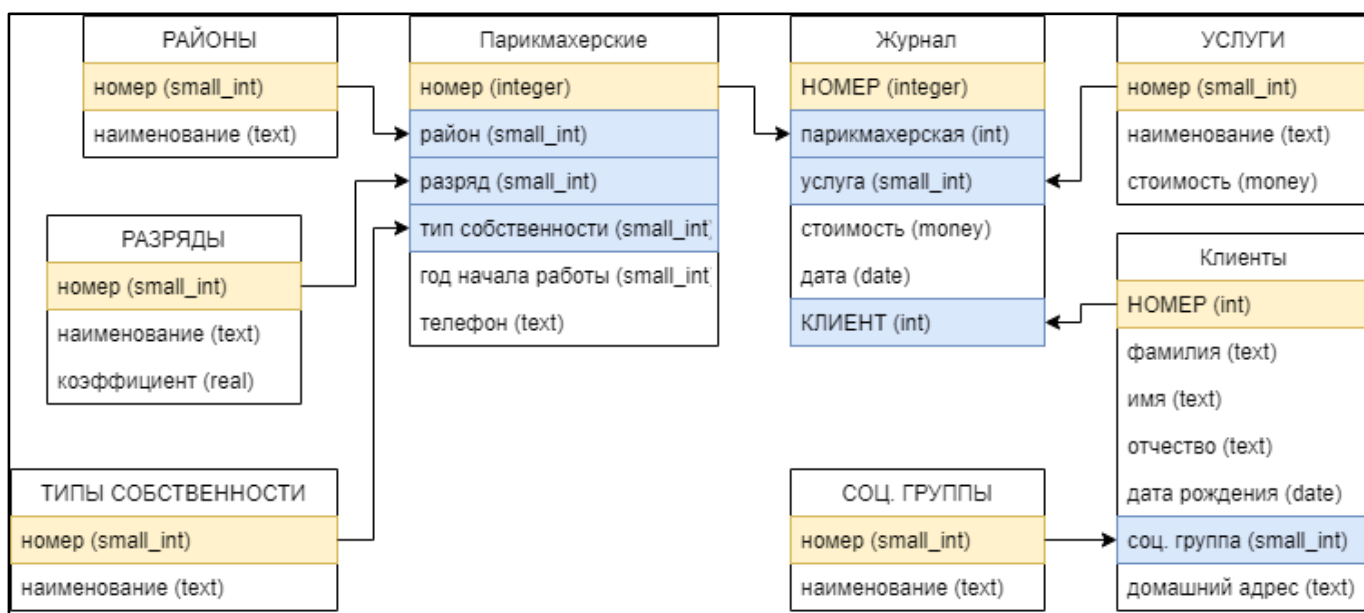


Рисунок 4.1 – Концептуальная модель базы данных «Парикмахерские»

Учитывая ожидаемое количество записей в таблице «Журнал», а также необходимость контроля доступа к данным, данной таблице будет введено внутреннее подразделение по дополнительному полю «Пользователь» при фактической реализации базы.

4.2 Создание таблиц, доменов, индексов, сиквенсов

Разрабатываемая БД состоит из трёх основных таблиц, отражающих контролируемые парикмахерские, их клиентов и журналы посещений, а также из пяти редко изменяемых таблиц-списков, к которым не имеют доступа обычные пользователи БД (см. рис. 4.2).

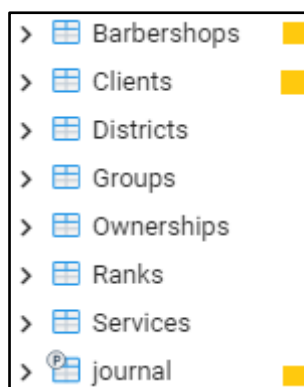


Рисунок 4.2 – Таблицы в БД «Парикмахерские»

Для данной БД не необходимости создавать специализированные домены.

Индексы созданы для каждой из основных таблиц по полям главных ключей, а также дополнительно по фамилии и имени клиента в таблице «Клиенты» и всем внешним ключам таблицы «Журнал» (см. рис. 4.3).

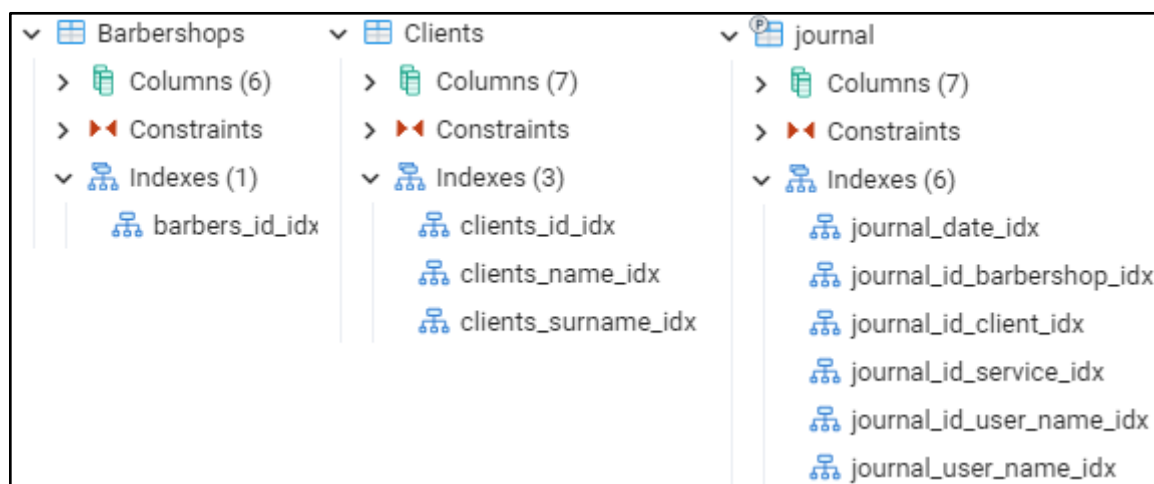


Рисунок 4.3 – Индексы БД

Сиквенсы обеспечивают последовательное повышение числового значения, благодаря чему отлично подходят для реализации идентификации записей таблиц.



Рисунок 4.4 – Сиквенсы БД

В БД было создано 8 сиквенсов, которые отвечают за последовательную выдачу значений полям «id» в таблицах при срабатывании триггеров перед добавлением записи (см. рис. 4.4).

4.3 Разработка триггеров

Исходя из поставленной техническим заданием задачи, для каждой таблицы были созданы простые триггеры перед добавлением, которые, используя соответствующие таблицам сиквенсы, присваивают в поле идентификатора значение, если оно не было задано.

Также созданы следующие триггеры:

- Триггер перед обновлением в таблице «Клиенты» – если происходит изменение в фамилии, имени или отчестве, к соответствующему значению дописывается маркер «(upd)» (см. рис. 4.5 - 4.6).

```

BEGIN
IF (OLD.surname != NEW.surname) THEN NEW.surname := NEW.surname || ' (upd)'; END IF;
IF (OLD.name != NEW.name) THEN NEW.name := NEW.name || ' (upd)'; END IF;
IF (OLD.fathername != NEW.fathername) THEN NEW.fathername := NEW.fathername || ' (upd)'; END IF;
RETURN NEW;
END;

```

Рисунок 4.5 – Код тела триггера перед обновлением

1000	Wilkinson		1000	Винкинсон (upd)
------	-----------	--	------	-----------------

Рисунок 4.6 – Пример работы триггера перед обновлением

- Триггер перед удалением в таблице «Клиенты» – передаёт все записи в журнале «нулевому» клиенту – записи, созданной для хранения прибыли от клиентов, регистрация которых была удалена (см. рис. 4.7).

```

BEGIN
UPDATE journal SET id_client=0 WHERE id_client=OLD.id;
RETURN OLD;
END;

```

Рисунок 4.7 – Код тела триггера перед удалением

- Триггер после добавления в таблице «Парикмахерские» – если происходит добавление записи о парикмахерской, дата открытия которой позже текущей (то есть её только предстоит открыть), то, независимо от указанных данных, ранг будет удалён (см. рис. 4.8).

```

BEGIN
IF (NEW."starting date" > CURRENT_DATE)
THEN UPDATE "Barbershops" SET rank = NULL WHERE id = NEW.id; END IF;
RETURN NEW;
END;

```

Рисунок 4.8 – Код тела триггера после добавления

- Триггеры после изменения и после удаления в таблице «Парикмахерские» – добавляют новую запись в таблицу «Парикмахерские» с указанными данными до взаимодействия, а также датой взаимодействия, пользователем, который производил взаимодействие и старым (и новым – при взаимодействии изменения) идентификатором записи (см. рис. 4.9 - 4.10).

```
BEGIN
INSERT INTO public."Barbershops" (district, rank, ownership, "starting date", phone, user_name)
VALUES (OLD.district, OLD.rank, OLD.ownership, OLD."starting date", OLD.phone,
        TO_CHAR(CURRENT_DATE, 'dd.mm.yyyy') || ' ' || CURRENT_USER || ' upd id:' || OLD.id || ' ' || NEW.id);
RETURN NEW;
END;
```

Рисунок 4.9 – Код тела триггера после изменения

```
BEGIN
INSERT INTO public."Barbershops" (district, rank, ownership, "starting date", phone, user_name)
VALUES (OLD.district, OLD.rank, OLD.ownership, OLD."starting date", OLD.phone,
        TO_CHAR(CURRENT_DATE, 'dd.mm.yyyy') || ' ' || CURRENT_USER || ' del id:' || OLD.id);
RETURN OLD;
END;
```

Рисунок 4.10 – Код тела триггера после удаления

Таким образом БД содержит восемь индексных триггеров и пять управляющих, соответствующих заданию.

4.4 Организация многоуровневого доступа к данным

Для работы с БД было создано три роли-группы пользователей: группа директоров, группа менеджеров и группа работников. Взаимодействие со структурой базы и записями таблиц-справочников доступно только для администратора БД. Для выполнения же работ, группам пользователей предоставлены необходимые и исчерпывающие разрешения на взаимодействие с объектами базы данных посредством созданных хранимых функций (см. пример на рис. 4.5) и представлений таблиц.

```

(=) avg_op_cost_as_services_agr_whr(service_n text)
(=) barbers_by_ownersh_in_key(o_name text)
(=) barbers_by_rank_in_key(r_name text)
(=) clients_by_birth_in_date(birth text)
(=) clients_on_group_slct_whr_slct(group_n text)
(=) get_parent_roles(user_name text)
(=) journal_by_date_in_date(op_date text)
(=) profit_from_xpsv_servs_on_date_agr_whr_hav(date_op text)
(=) ranks_at_barbers_ro(b_rank text)

```

Рисунок 4.11 – Функции БД, созданные на основе запросов, указанных в техническом задании (запросы описаны в п.п. 4.7)

Также следует отметить, что таблица «Журнал» разделена на партии, доступ к которым имеют лишь пользователи, которым они принадлежат.

4.5 Разграничение доступа к данным на уровне строк

Разграничение доступа к данным на уровне строк обеспечивает контроль воздействия пользователей, скрывая записи по ограничениям, что позволяет создать механизм влияния пользователей только на записи, созданные ими.

Для выполнения данной задачи в основные таблицы, к которым производится доступ пользователя включены дополнительные поля, хранящие имена пользователей, которые создают записи.

Политики контроля строк добавлены во все основные таблицы, кроме таблицы «Парикмахерские», так как она доступна только пользователям группы директоров, каждый из которых может управлять всеми записями данной таблицы (см. рис. 4.12).

```

CREATE POLICY users_policy
ON public.journal
AS PERMISSIVE
FOR ALL
TO public
USING ((user_name = CURRENT_USER));

```

Рисунок 4.12 – Политика контроля в таблице «Журнал»

4.6 Партицирование одной из основных таблиц БД

При разработке БД проведено партицирование таблицы «Журнал» по полю «Пользователь», отражающему того пользователя, который добавлял запись в таблицу, которое также используется и для обеспечения безопасности на уровне строк. Подобное разделение позволит ускорить взаимодействие пользователей-работников с созданными ими данными (см. рис. 4.13-4.14).

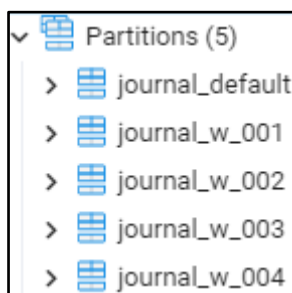


Рисунок 4.13 – Партии таблицы «Журнал»

```
CREATE TABLE public.journal_w_001 PARTITION OF public.journal
FOR VALUES IN ('w_001')
```

Рисунок 4.14 – Создание партии первого работника в таблице «Журнал»

Создание партии происходит сразу после создания новой роли работника, при удалении пользователя все данные из соответствующей ему партии перемещаются в стандартную партию. Данный процесс обеспечивается следующим запросом, образуемом на сервере:

```
CREATE ROLE {arguments[0]} WITH LOGIN NOSUPERUSER NOCREATEDB NOCREATEROLE INHERIT NOREPLICATION
CONNECTION LIMIT 1 PASSWORD '{arguments[1]}';
GRANT worker_gp TO {arguments[0]};
CREATE TABLE journal_{arguments[0]} PARTITION OF journal FOR VALUES IN ('{arguments[0]}')
```

4.7 Проектирование запросов к базе данных

Исходя из поставленной задачи, было создано 20 запросов, из которых 10 – простые запросы, и ещё 10 – сложные или итоговые запросы. Образцы созданных запросов представлены далее:

- Четыре запроса, использующих внутреннее соединение с условиями по внешнему ключу или по дате:

- = запрос, выбирающий информацию о парикмахерских с указанным правом собственности (см. рис. 4.15-4.16);

- = запрос, выбирающий парикмахерские указанного ранга;

- = запрос, выбирающий клиентов старше указанной даты рождения;

- = запрос, выбирающий записи из журнала за указанную дату.

```
SELECT b.id, o.name, TO_CHAR(b."starting date", 'dd.mm.yyyy'), d.name
FROM "Barbershops" AS b
INNER JOIN "Ownerships" AS o ON o.id = b.ownership
INNER JOIN "Districts" AS d ON d.id = b.district
WHERE o.name = o_name
ORDER BY b.id, b."starting date", d.name
```

Рисунок 4.15 – Код одного из запросов, использующих внутреннее соединение с условиями

barber integer	ownersh text	start date text	district text
12	государственная	28.10.2018	Куйбышевский
14	государственная	09.12.2009	Киевский
24	государственная	22.07.2013	Куйбышевский
25	государственная	16.09.2007	Киевский
28	государственная	26.03.2018	Куйбышевский

Рисунок 4.16 – Результат выполнения одного из запросов, использующих внутреннее соединение с условиями

- Три запроса, использующих внутреннее соединение без условия:
- = запрос, выводящий информацию о парикмахерских (см. рис. 4.17 - 4.18);
- = запрос, выводящий информацию о клиентах;
- = запрос, выводящий информацию из журнала о заказах.

```
SELECT b.id,
       o.name AS ownership,
       r.name AS rank
FROM "Barbershops" b
     JOIN "Ranks" r ON r.id = b.rank
     JOIN "Ownerships" o ON o.id = b.ownership
ORDER BY r.id, o.name, b.id;
```

Рисунок 4.17 – Код одного из запросов, использующих внутреннее соединение без условий

id integer	ownership text	rank text
5	акционерная	Второй
21	акционерная	Второй
36	акционерная	Второй
37	акционерная	Второй
40	акционерная	Второй
49	акционерная	Второй

Рисунок 4.18 – Результат выполнения одного из запросов, использующих внутреннее соединение без условий

- Запрос, использующий левое внешнее соединение, выводящий информацию о безранговых парикмахерских (см. рис. 4.19 - 4.20).

```
SELECT b.id,
       to_char(b."starting date"::timestamp with time zone, 'dd.mm.yyyy'::text) AS to_char,
       d.name
FROM "Barbershops" b
     LEFT JOIN "Ranks" r ON r.id = b.rank
     LEFT JOIN "Districts" d ON d.id = b.district
WHERE b.rank IS NULL
ORDER BY d.name, b."starting date";
```

Рисунок 4.19 – Код запроса, использующего левое внешнее соединение

id integer	to_char text	name text
2070	31.07.2005	Ворошиловский
2081	19.10.2005	Ворошиловский
2031	12.05.2013	Ворошиловский
2029	09.10.2014	Ворошиловский
2030	18.07.2015	Ворошиловский
2065	04.03.2017	Ворошиловский

Рисунок 4.20 – Результат выполнения запроса, использующего левое внешнее соединение

- Запрос, использующий правое внешнее соединение, выводющий парикмахерские указанного ранга, а также коэффициенты стоимости услуг остальных рангов (см. рис. 4.21 - 4.22).

```
SELECT r.name, r.factor, b.phone, TO_CHAR(b."starting date", 'dd.mm.yyyy')
FROM "Barbershops" AS b
RIGHT OUTER JOIN "Ranks" AS r ON r.id = b.rank AND r.name = b_rank
ORDER BY b."starting date", r.factor
```

Рисунок 4.21 – Код запроса, использующего правое внешнее соединение

rank text	factor real	phone text	starting date text
Первый	1	071-749-51-59	21.01.2005
Первый	1	071-749-51-59	21.01.2005
Первый	1	071-749-51-59	21.01.2005
Первый	1	071-749-51-59	21.01.2005
Первый	1	071-513-79-93	13.03.2005
Первый	1	071-513-79-93	13.03.2005
Первый	1	071-513-79-93	13.03.2005
Первый	1	071-513-79-93	13.03.2005
Первый	1	071-732-33-31	07.04.2005
Первый	1	071-732-33-31	07.04.2005
Первый	1	071-732-33-31	07.04.2005

Рисунок 4.22 – Результат выполнения запроса, использующего правое внешнее соединение

- Запрос на запросе, созданном по принципу левого внешнего соединения, выводит районы города, в которых есть парикмахерские (см. рис. 4.23 - 4.24).

```
SELECT DISTINCT barbers.district
FROM ( SELECT b.id,
      to_char(b."starting date"::timestamp with time zone, 'dd.mm.yyyy'::text) AS s_date,
      d.name AS district
FROM "Barbershops" b
  LEFT JOIN "Ranks" r ON r.id = b.rank
  LEFT JOIN "Districts" d ON d.id = b.district
WHERE b.rank IS NULL
ORDER BY d.name, b."starting date") barbers;
```

Рисунок 4.23 – Код запроса на запросе, созданном по принципу левого внешнего

district
text
Ворошиловский
Калининский
Киевский
Кировский
Куйбышевский
Ленинский
Петровский

Рисунок 4.24 – Результат выполнения запроса на запросе, созданном по принципу левого внешнего

- Четыре итоговых запроса:

= запрос с условием по группам, выводящий информацию о прибыли и количестве заказов по услугам, прибыль от которых превзошла 250000 условных единиц валюты (см. пример на рис. 4.25 - 4.26);

= запрос, выводящий информацию о количестве клиентов различных групп;

= запрос, выводящий информацию о средней оплате за указанную услугу;

= запрос, выводящий информацию о прибыли за указанную дату от наиболее дорогих услуг;

```

SELECT count(j.id) AS operations,
       sum(j.cost) AS sum,
       s.name
FROM journal j
     JOIN "Services" s ON s.id = j.id_service
GROUP BY s.name
HAVING sum(j.cost) > 250000::money
ORDER BY s.name, (count(j.id));

```

Рисунок 4.25 – Код одного из итоговых запросов

operations bigint	sum money	name text
1689	347 340,00 ?	Маникюр
912	471 600,00 ?	Массаж
1760	633 920,00 ?	Педикюр
1751	535 380,00 ?	Покраска
1897	291 360,00 ?	Укладка

Рисунок 4.26 – Результат выполнения одного из итоговых запросов

- Запрос на запросе по принципу итогового запроса, выводящий информацию о количестве парикмахерских с различными коэффициентами стоимости услуг (см. рис. 4.27 - 4.28).

```

SELECT ( SELECT count(b.id) AS count
         FROM "Barbershops" b
         JOIN "Ranks" r ON r.id = b.rank
         WHERE b.rank = rnk.id) AS count,
       rnk.factor
FROM "Ranks" rnk
ORDER BY (( SELECT count(b.id) AS count
            FROM "Barbershops" b
            JOIN "Ranks" r ON r.id = b.rank
            WHERE b.rank = rnk.id)) DESC, rnk.factor;

```

Рисунок 4.27 – Код запроса на запросе, созданном по принципу итогового

count bigint	factor real
966	1
556	0.9
512	1.2
0	0.6
0	0.7

Рисунок 4.28 – Результат выполнения запроса на запросе,
созданном по принципу итогового

- Запрос с использованием объединения, выводящий информацию о количестве парикмахерских по районам и их общее количество (см. рис. 4.29 - 4.30).

```
SELECT count("Barbershops".id)::text AS field_1,
       '_Всего'::text AS field_2
  FROM "Barbershops"
UNION
SELECT count(b.id)::text AS field_1,
       d.name AS field_2
  FROM "Barbershops" b
       JOIN "Districts" d ON d.id = b.district
 GROUP BY d.name
 ORDER BY 2, 1;
```

Рисунок 4.29 – Код запроса с использованием объединения

2095	_Всего
102	Будёновский
272	Ворошиловский
263	Калининский
257	Киевский
262	Кировский
266	Куйбышевский
307	Ленинский
246	Петровский
120	Пролетарский

Рисунок 4.30 – Результат выполнения запроса с использованием объединения

- Запрос с подзапросом (с использованием in, not in, case, операциями над итоговыми данными), выводящий информацию о клиентах указанной группы (см. рис. 4.31 - 4.32).

```
SELECT c.surname, c.address
FROM "Clients" AS c, "Groups" AS g
WHERE g.id = c.group AND c.group IN
      (SELECT id FROM "Groups" WHERE name LIKE group_n)
ORDER BY 2, 1
```

Рисунок 4.31 – Код запроса с подзапросом

surname text	group text
Blackwell	107-4578 Nec Road
Blackwell	107-4578 Nec Road
Willis	116-3624 Auctor Avenue
Willis	116-3624 Auctor Avenue
Frye	131 Mollis. Road
Frye	131 Mollis. Road
Little	140-1684 Tellus. St.
Little	140-1684 Tellus. St.
Mcgee	1411 Nibh. St.
Mcgee	1411 Nibh. St.
House	160-8189 Mauris, Street
House	160-8189 Mauris, Street
Mcfarland	172-2527 Arcu. St.
Mcfarland	172-2527 Arcu. St.

Рисунок 4.32 – Результат выполнения запроса с подзапросом

4.8 Создание представлений и хранимых процедур, функций

Все созданные запросы, которые требуют ввода конкретизирующих данных от пользователя, оформлены в виде функций, другие, что собирают необходимую информацию из таблиц и сразу передают её, – в виде представлений.

Также создано модифицируемое представление, которое полностью отражает таблицу «Клиенты», исключая из выбранных строк содержащую «нулевого» клиента. Модификация таблицы клиентов через данное представление происходит при помощи триггеров вместо взаимодействия, которые применяют данные взаимодействия для вызова соответствующего взаимодействия с родительской таблицей (см. рис. 4.33 - 4.34).

```
SELECT * FROM "Clients" WHERE "Clients".id > 0 ORDER BY 1;
```

Рисунок 4.33 – Код тела модифицируемого представления

```
BEGIN
INSERT INTO "Clients"(surname, name, fathename, birthday, "group", address, user_name)
VALUES (NEW.surname, NEW.name, NEW.fathename, NEW.birthday, NEW."group", NEW.address, CURRENT_USER);
RETURN NEW;
END;

BEGIN
UPDATE "Clients" SET id=NEW.id, surname=NEW.surname, name=NEW.name,
fathename=NEW.fathename, birthday=NEW.birthday, "group"=NEW."group",
address=NEW.address, user_name=NEW.user_name
WHERE id=OLD.id;
RETURN NEW;
END;

BEGIN
DELETE FROM "Clients" WHERE id=OLD.id;
RETURN OLD;
END;
```

Рисунок 4.34 – Коды тел триггеров модифицируемого представления

Все функции, хранимые в базе данных, отвечающие за выборку данных, отмечены как стабильные («STABLE»), поскольку не влияют на содержимое базы. Те же функции, которые применяются для внесения или изменения записей отмечены как изменчивые («VOLETILE»). Каждая из функций-обёрток разработанных запросов на выборку принимает текстовое значение, используемое для отбора выбираемых записей, и возвращает таблицу выбранных записей (см. пример на рис. 4.35).

```
CREATE OR REPLACE FUNCTION public.c13_3_fields(birth text)
  RETURNS TABLE(field_1 text, field_2 text, field_3 text)
  LANGUAGE 'sql' STABLE

AS $BODY$
SELECT g.name, c.surname, TO_CHAR(c.birthdate, 'dd.mm.yyyy')::text
  FROM "Clients" AS c
  INNER JOIN "Groups" AS g ON g.id = c.group
  WHERE c.birthdate < birth::date
  ORDER BY 1, 2
$BODY$;
```

Рисунок 4.35 – Функция запроса получения записей о клиентах, которые старше указанной даты

Из дополнительных функций, обеспечивающих работу системы созданы:

- Функция, выдающая родительские роли указанного пользователя, используемая для определения группы, к которой относится авторизуемый пользователь. На вход функции передаётся текстовое значение – пользовательское имя, относительно которого будет вестись поиск. Возвращает функция таблицу с одним текстовым полем – перечнем ролей указанного пользователя (см. рис. 4.36).

```
CREATE OR REPLACE FUNCTION get_parent_roles(user_name text)
  RETURNS TABLE(field_1 text)
  LANGUAGE 'sql' STABLE

AS $BODY$
SELECT r1.rolname::text FROM pg_roles AS r
LEFT JOIN pg_auth_members AS m ON (m.member = r.oid)
LEFT JOIN pg_roles AS r1 ON (r1.oid = m.roleid)
WHERE r.rolcanlogin AND r.rolname = user_name;
$BODY$;
```

Рисунок 4.36 – Функция получения родительских ролей пользователя

- Функция вывода всех пользователей группы «Работник», необходимая для работы пользователя-менеджера. Похожа на предыдущую функцию, но не имеет входных аргументов. Функция возвращает таблицу с текстовым полем, содержащим перечень пользователей-работников (см. рис. 4.37).

```
CREATE OR REPLACE FUNCTION public.c61_1_fields()
    RETURNS TABLE(field_1 text)
    LANGUAGE 'sql' STABLE

AS $BODY$
SELECT r.rolname::text FROM pg_roles AS r
LEFT JOIN pg_auth_members AS m ON (m.member = r.oid)
LEFT JOIN pg_roles AS r1 ON (r1.oid = m.roleid)
WHERE r.rolcanlogin AND r1.rolname = 'worker_gp';
$BODY$;
```

Рисунок 4.37 – Функция получения пользователей-работников

- Функции форматированного предоставления информации из таблиц «Клиенты» и «Парикмахерские», используемые при подготовке выпадающих списков, применяемых при создании и обновлении записей о заказах. Обе функции не имеют входных атрибутов, поскольку представляют из себя функции выборки, а также возвращают два текстовых поля – одно, содержащее идентификаторы записей, второе – содержащее текст, составленный из полей основной таблицы, наиболее точно и лаконично описывающих запись (см. рис. 4.38).

```
CREATE OR REPLACE FUNCTION public.c72_2_fields()
    RETURNS TABLE(field_1 text, field_2 text)
    LANGUAGE 'sql' STABLE

AS $BODY$
SELECT id, surname || ' ' || name FROM "Clients_v" ORDER BY 2;
$BODY$;
```

```
CREATE OR REPLACE FUNCTION public.c72_2_fields()
    RETURNS TABLE(field_1 text, field_2 text)
    LANGUAGE 'sql' STABLE

AS $BODY$
SELECT b.id, d.name || ' ' || r.name FROM "Barbershops" b
INNER JOIN "Districts" d ON d.id = b.district
INNER JOIN "Ranks" r ON r.id = b.rank ORDER BY 2;
$BODY$;
```

Рисунок 4.38 – Функции получения данных

из таблиц «Клиенты» и «Парикмахерские»

5 РАЗРАБОТКА КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

Работа с клиентским приложением для любого из пользователей начинается с общей для каждого формы авторизации, представленной на рисунке 5.1.

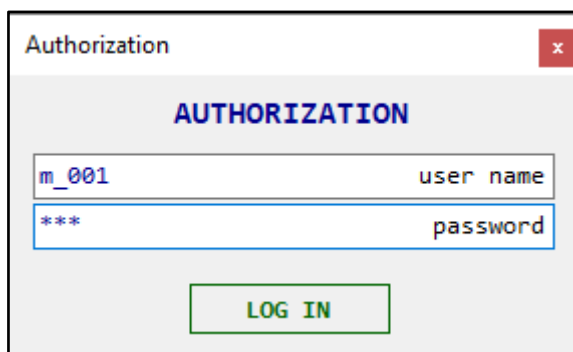
A screenshot of a window titled "Authorization" with a red close button in the top right corner. The window has a light gray background. In the center, the word "AUTHORIZATION" is written in blue, bold, uppercase letters. Below it, there are two input fields. The first field contains the text "m_001" and is labeled "user name" on the right. The second field contains three asterisks "***" and is labeled "password" on the right. Below these fields is a green rectangular button with the text "LOG IN" in green, bold, uppercase letters.

Рисунок 5.1 – Форма авторизации

После авторизации пользователя встречает главная форма клиентского приложения. Её составляющие – это поле имени отображаемой таблицы, область данных, в которую выводятся данные в виде таблицы, поисковик, позволяющий найти все записи, содержащие какое-либо значение, и счётчик страниц (см. рис. 5.2).

A screenshot of a window titled "DB client" with a red close button in the top right corner. The window has a light gray background. At the top, there is a text input field for the table name, followed by a blue button labeled "MENU". The main area of the window is a large gray rectangle representing the data table. At the bottom, there is a search bar with a red "X" button and a blue "SEARCH" button. To the right of the search bar, the word "PAGE" is displayed, followed by a text input field containing the number "1" and a small up/down arrow button.

Рисунок 5.2 – Главная форма клиентского приложения

Для любого из пользователей главная форма остаётся одинаковой, и изменения претерпевает только область данных, которая автоматически разбивается на необходимое количество столбцов и заполняется информацией при нажатии на соответствующие управляющие элементы – пункты меню или список действий при нажатии правой клавиши мыши на области данных.

5.1 Формы и компоненты для работы в роли «Менеджер»

Существенную разницу в пользовательском интерфейсе можно заметить в форме меню – отдельно открываемом окне при помощи соответствующей кнопки на главной форме.

Каждая из панелей меню содержит кнопку выхода из аккаунта, а также список доступных представлений данных. Так, вид меню менеджера представлен на рисунке 5.3.

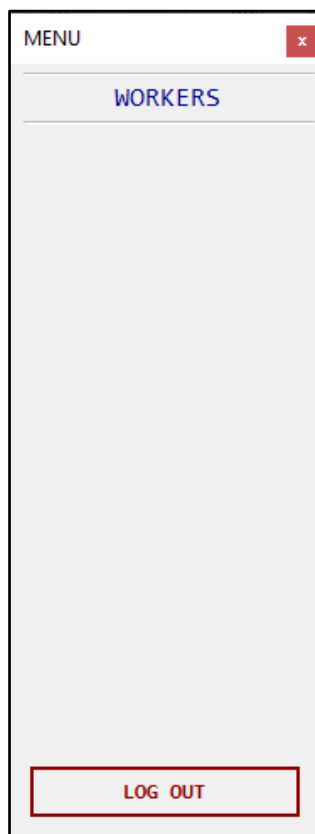


Рисунок 5.3 – Форма-меню менеджера

Для создания и обновления записей в базе данных применяется специально разработанный класс, создающий диалоговую форму с изменяемым количеством текстовым полей, которым возможно установить заполнитель, а также каждое из полей при определённых условиях может быть создано как выпадающий список вместо текстовой области.

В случае с функционалом менеджера, создаются диалоги добавления и изменения работника (см. рис. 5.4).

Рисунок 5.4 – Форма добавления работника

5.2 Формы и компоненты для работы в роли «Директор»

После авторизации директора ему доступен просмотр записей о парикмахерских города и список различных отчётов о парикмахерских и доходах (см. рис. 5.5).

district	rank	ownership	starting date	phone
Будёновский	Второй	частная		
Киевский	Второй	частная		
Киевский	Первый	акционер		
Куйбышевский	Первый	акционер		
Ворошиловский	Второй	акционер		
Кировский	Второй	частная		
Куйбышевский	Высший	иностран		
Куйбышевский	Первый	иностран		
Пролетарский	Первый	частная		
Будёновский	Первый	частная		
Будёновский	Второй	иностранная	2015-11-02	071-574-34-78
Куйбышевский	Первый	государственная	2018-10-28	071-284-52-45
Будёновский	Первый	иностранная	2023-11-19	071-529-21-35
Киевский	Первый	государственная	2009-12-09	071-559-71-29
Киевский	Высший	иностранная	2024-01-06	071-727-25-26
Калининский	Первый	акционерная	2005-03-13	071-513-79-93
Ленинский	Первый	частная	2007-09-09	071-485-44-02

Рисунок 5.5 – Основные формы для работы директора

Удаление и изменение записей поддерживает множественный выбор, а также, прежде чем выполнить удаление, пользователю будет предложено выполнить подтверждение во избежание потери данных, что продемонстрировано на рисунке 5.6.

Barbershops					MENU
district	rank	ownership	starting date	phone	
Будёновский	Второй	частная		0710000001	
Киевский	Второй	частная		071-187-98-51	
Киевский	Первый	акционер		071-565-87-24	
Куйбышевский	Первый	акционер		071-613-16-88	
Ворошиловский	Второй	акционер		071-118-88-44	
Кировский	Второй	частная		071-802-38-58	
Куйбышевский	Высший	иностранная	2007-08-29	071-523-75-50	

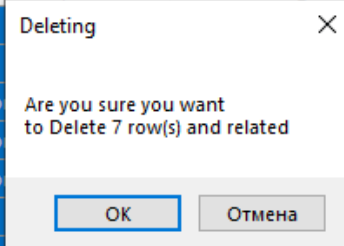


Рисунок 5.6 – Диалоговое окно удаления записей

5.3 Формы и компоненты для работы в роли «Работник»

Меню работников позволяет просмотреть записи о заказах, которые были сделаны данным работником, о клиентах, а также связанные отчёты (см. рис. 5.7).

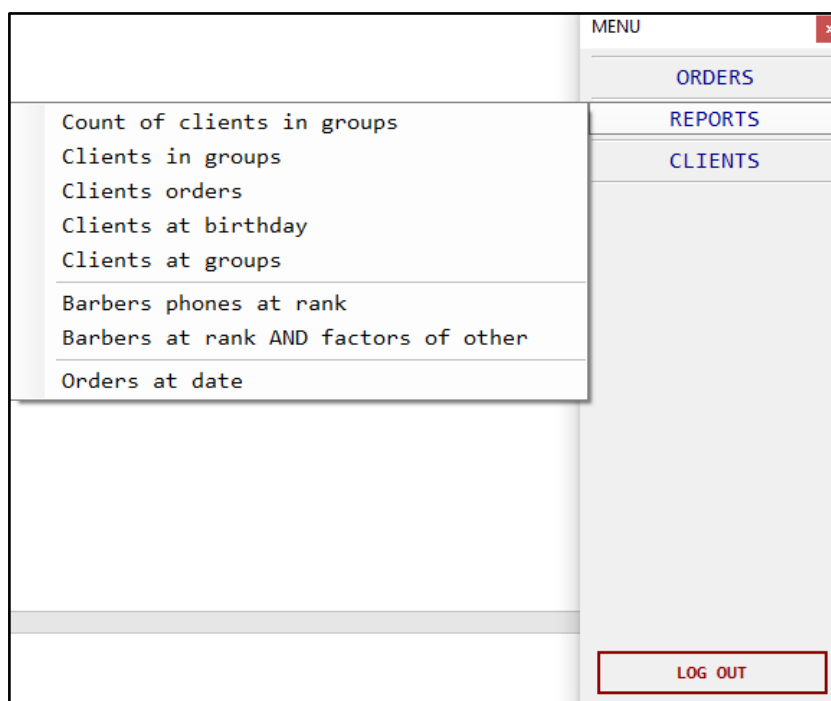


Рисунок 5.7 – Форма-меню работника

Особенностью работы клиентского приложения при пользователе-работнике можно назвать дополнительную форму с информацией о группах клиентов, содержащую область табличных данных, схожую с областью основной формы по функционалу, обеспечивающую принцип работы составной формы (см. рис. 5.8).

CLIENT GROUPs
group
Предприниматель
Банкир
Инженер
Разнорабочий
Юрист
Госслужащий
Преподаватель
Учёный

Рисунок 5.8 – Форма групп клиентов

5.4 Генерация результатов не менее трех итоговых запросов (диаграммы, экспорт в Excell)

В клиентском приложении реализован доступ ко всем разработанным итоговым запросам, которые представлены также, как и любой другой вывод данных – в виде пункта меню для вызова и таблицы при отображении результата. Так, результат итогового запроса, демонстрирующего количество парикмахерских в районах города, а также их общее количество, показан на рисунке 5.9.

Count of barbers at districts		MENU
count of Barbers	district	
2097	_Всего	
102	Будёновский	
272	Ворошиловский	
263	Калининский	
257	Киевский	
262	Кировский	
268	Куйбышевский	
307	Ленинский	
246	Петровский	
120	Пролетарский	

Рисунок 5.9 – Результат итогового запроса

Форма с диаграммой, которую можно вызвать при просмотре отчёта о количестве клиентов в группах – дополнительный способ визуализации данных, доступных пользователям-работникам. Также на ней расположена кнопка, позволяющая сохранить данный отчёт в Excel-формате (см. рис. 5.10).

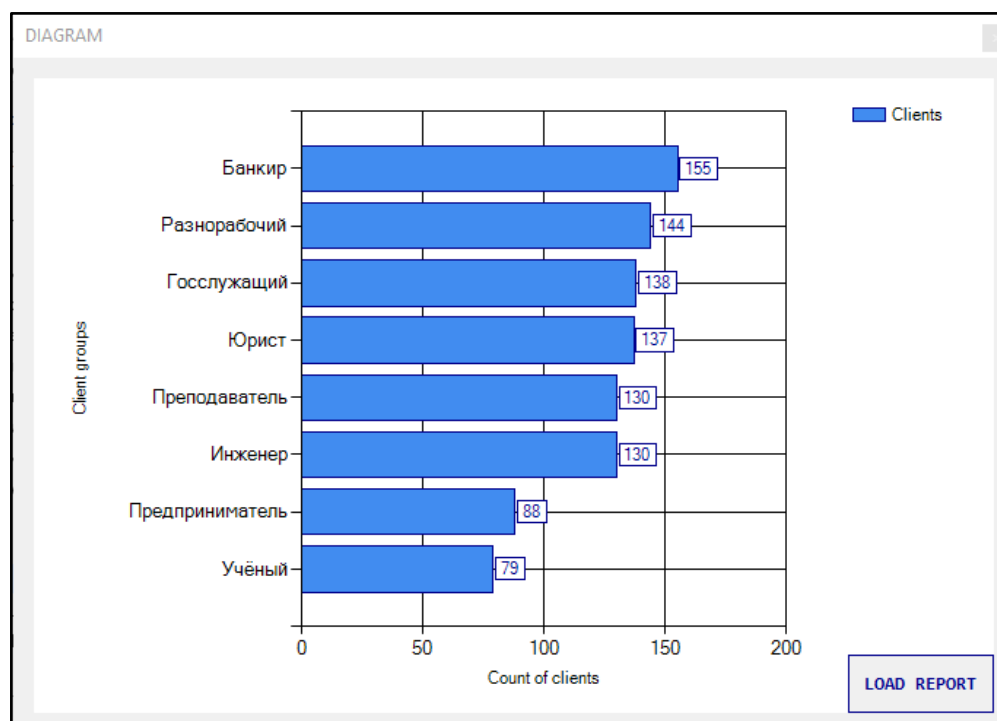


Рисунок 5.10 – Форма с диаграммой

6 ТЕСТИРОВАНИЕ РАЗРАБОТАННОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ

В первую очередь, безопасность работы с информационной системой обеспечивается представленной ранее формой авторизации, функционирование которой основано на подключении к базе данных за счёт различных пользователей.

Одновременная работа нескольких пользователей обеспечивается средствами сервера PostgreSQL, а также разработанного серверного приложения – прослойки между клиентами системы и базой данных.

Также все операции над данными проводятся транзакциями, т.е. либо выполняются полностью, либо не выполняются вовсе, что обеспечивает сохранность базы данных при появлении сбоев в системе.

Для тестирования разработанной клиент-серверной системы в базу данных было внесено более десяти тысяч сгенерированных записей, распределённых между различными таблицами, а также проведено ручное пользовательское тестирование всех доступных функций приложений.

Все выявленные в ходе тестирования недочёты в структуре и реализации базы данных и приложений устранены в ходе доработки проекта.

ВЫВОДЫ

В ходе выполнения курсовой работы получены практические навыки проектирования, моделирования и создания баз данных средствами СУБД PostgreSQL, а также разработки клиент-серверных систем контроля баз данных, основанных на многопользовательском доступе с использованием различных ролей.

В результате выполнения курсовой работы было проведено инфологическое, даталогическое и физическое проектирование модели базы данных по заданному варианту (информационная система «Парикмахерские»), был создан проект базы данных, созданы и заполнены соответствующим образом все необходимые таблицы (таблицы-справочники, вспомогательные таблицы и основные таблицы).

Также были разработаны серверное и клиентское приложения на языке программирования высокого уровня C#, обеспечивающие взаимодействие пользователя с базой данных. В программе реализована возможность сохранения результатов запросов в формат Excel.

Разработанная система создана в целях получения и закрепления навыков создания клиент-серверной информационной системы средствами СУБД и не подлежит к практическому применению, кроме демонстрации возможностей при моделировании и реализации схожих проектов во время образовательного процесса.

Для пользователей разработанной системы было написано соответствующие руководства использования.

СПИСОК ЛИТЕРАТУРЫ

1. Базы данных. [Электронный ресурс]. URL: <http://flash-library.narod.ru/Ch-Informatics/lektion/lektion7.html>.
2. Системы управления базами данных (СУБД). [Электронный ресурс]. URL: [http://wiki.mvtom.ru/index.php/Системы_управления_базами_данных_\(СУБД\)](http://wiki.mvtom.ru/index.php/Системы_управления_базами_данных_(СУБД)).
3. Шилдт, Герберт. С# 4.0: полное руководство. — М.: Вильямс, 2011. — 1056 с.
4. Суркова Н.Е., Остроух А.В. Методология структурного проектирования информационных систем: Монография — Красноярск.
5. Моргунов, Е. П. PostgreSQL. Основы языка SQL: учеб. пособие / Е. П. Моргунов; под ред. Е. В. Рогова, П. В. Лузанова. — СПб.: БХВ-Петербург, 2018. — 336 с.: ил. Научно-инновационный центр, 2014. — 190 с.
6. Бьюли А. Изучаем SQL. — М.: Вильямс, 2010. — 100 с.

ПРИЛОЖЕНИЕ А.
Техническое задание

ГОУ ВПО «Донецкий национальный технический университет»
Факультет Интеллектуальных систем и программирования
Кафедра "Программная инженерия" им. Л.П. Фельдмана

Утверждаю
Зори С.А.

08.02.2022 г.

ТЕХНИЧЕСКОЕ ЗАДАНИЕ
на курсовую работу по дисциплине
«Программирование систем с серверами баз данных»

выдано студенту группы ПИ-19 «Б» Носаченко Артёму Александровичу

Тема: «Создание клиент-серверной информационной системы средствами СУБД»

Описание предметной области:

14. Для учета работы парикмахерских города необходима информация о парикмахерских (номер, район города, разряд (высший, первый, второй), тип собственности (частная, государственная, акционерная,...), год начала функционирования, телефон), клиентах (ФИО, дата рождения, социальная группа (предприниматель, банковский служащий, инженер, рабочий,...), домашний адрес)) и оказанных клиентам услугах (парикмахерская, название услуги (стрижка, завивка, укладка, маникюр, массаж,...), стоимость (зависит от разряда парикмахерской), дата оказания услуги).

Задание на курсовую работу

1. Спроектировать концептуальную модель базы данных (БД) для заданной предметной области и представить ее в виде взаимосвязанных таблиц, находящихся в третьей нормальной форме (в случае денормализации БД – обосновать необходимость). Выделить базовые таблицы и таблицы-справочники, указать для них первичные и внешние ключи.
2. Создать базу данных в среде СУБД средствами языка SQL. Добавить таблицы, домены, индексы.
3. Разработать не менее шести триггеров (по одному для каждого типа события), как минимум для двух различных таблиц БД. Триггеры типа BEFORE INSERT должны быть созданы для всех таблиц и с использованием генераторов задавать значение первичного ключа для вновь добавляемой записи.
4. Заполнить таблицы БД с использованием соответствующих запросов на языке SQL (не менее десяти записей в каждом справочнике, не менее 10 000 - 50 000 псевдослучайных записей в таблицах).
5. Сформулировать следующие виды запросов:
 - симметричное внутреннее соединение с условием (два запроса с условием отбора по внешнему ключу, два – по датам);
 - симметричное внутреннее соединение без условия (три запроса);
 - левое внешнее соединение;
 - правое внешнее соединение;
 - запрос на запросе по принципу левого соединения;
 - итоговый запрос без условия;
 - итоговый запрос без условия с итоговыми данными вида: «всего», «в том числе»;
 - итоговые запросы с условием на данные (по значению, по маске, с использованием индекса, без использования индекса);
 - итоговый запрос с условием на группы;
 - итоговый запрос с условием на данные и на группы;
 - запрос на запросе по принципу итогового запроса;
 - запрос с использованием объединения
 - запросы с подзапросами (с использованием in, not in, case, операциями над итоговыми данными).
6. Запросы без параметров реализовать в виде представлений, остальные запросы – в виде хранимых процедур и/или функций. Создать, по меньшей мере, одно модифицируемое представление, используя механизм триггеров. Вся логика проектируемого ПО – на сервере.

7. Разработать клиентское приложение, которое предоставляет следующие возможности для работы с созданной базой данных:

- многопользовательский режим работы (одна программа для всех ролей – ситуативный доступ к интерфейсу)
- наличие нескольких ролей пользователя (менеджер – добавление/удаление/редактирование пользователей, их прав/ролей; директор – просмотр отчётов о прибыли и убытках, работник – создание записей о проводимых работах, изучение личного дохода)
- просмотр содержимого таблиц и представлений (здесь и далее – с учетом прав пользователей);
- добавление, редактирование и удаление записей таблиц и модифицируемых представлений;
- работа с наборами данных, находящимися в отношении «один-ко-многим» (создать составную форму для просмотра и редактирования данных родительской и дочерней таблиц);
- поиск и фильтрация данных отображаемых таблиц;
- просмотр результатов выполнения запросов;
- визуализация результатов одного из итоговых запросов (диаграммы, экспорт в Excel).

8. Обеспечить защиту данных, информации от несанкционированного доступа, сделать защиту на уровне строк, выполнить партиционирование одной из основных таблиц

Рекомендуемое содержание пояснительной записки

Титульный лист

Реферат

Содержание

Введение

1. Описание предметной области, постановка задачи

2. Обоснование выбора СУБД, описание возможностей СУБД

3. Обоснование выбора инструментальных средств для написания

клиентской части, проектирование структуры ПО

3.1 Невизуальные компоненты для работы с данными

3.2 Визуальные компоненты отображения данных

3.3 Разработка шаблонов приложений для работы с таблицами базы данных

4. Проектирование базы данных в выбранной СУБД

4.1 Проектирование концептуальной модели БД

4.2 Создание таблиц, доменов, индексов, сиквенсов

- 4.3 Разработка триггеров
- 4.4 Организация многоуровневого доступа к данным
- 4.5 Разграничение доступа к данным на уровне строк (в зависимости от роли и логина)
- 4.6 Партицирование одной из основных таблиц БД
- 4.7 Проектирование запросов к базе данных
- 4.8 Создание представлений и хранимых процедур, функций
- 5. Разработка клиентского приложения
 - 5.1 Формы и компоненты для работы в роли «Менеджер»
 - 5.2 Формы и компоненты для работы в роли «Директор»
 - 5.3 Формы и компоненты для работы в роли «Работник»
 - 5.4 Генерация результатов не менее трех итоговых запросов (диаграммы, экспорт в Excell)
- 6 Тестирование разработанной информационной системы (в т.ч. включая защиту от несанкционированного доступа, одновременную работы с данными, каскадное удаление)
 - Заключение/выводы и предложения
 - Список литературы
 - Приложение А. Техническое задание
 - Приложение Б. Листинг шаблонов
 - Приложение В. Листинг серверного приложения
 - Приложение Г. Листинг клиентского приложения
 - Приложение Д. Руководство пользователя
 - Приложение Е. Руководство суперпользователя
 - Приложение Ж. Руководство администратора

График выполнения курсовой работы

Неделя	Работа
1-2	Выдача и изучение задания
3	Анализ требований к системе и способов их реализации
4-5	Проектирование и реализация БД (таблицы, домены, индексы, роли, RLS, партиционирование)
6-7	Создание триггеров и заполнение таблиц БД
8-9	Создание представлений и хранимых процедур, запросов
10-13	Разработка клиентского приложения
14	Тестирование и отладка системы
15	Оформление пояснительной записки
16-17	Защита курсовой работы

Дата выдачи задания

08.02.2022 г.

Задание принял



Носаченко А. А.

Руководители проекта

Щедрин С. В.

Ногтев Е. А.

ПРИЛОЖЕНИЕ Б.

Листинг серверного приложения

Основной класс сервера:

```
class Server
{
    private static int timeout = 5 * 60 * 1000; // todo [:]
    RECEIVE TIMEOUT [ sec * 1000 ]
    private static int max_count = 5;      // todo [:] MAX
    USERS COUNT
    private static int counter = 0;
    private static void DecreaseCounter() { counter--; }

    private const int port = 8888;
    private static TcpListener listener;

    private static void Main(string[] args)
    {
        try
        {
            listener = new
            TcpListener(IPAddress.Parse("127.0.0.1"), port);
            listener.Start();

            Console.WriteLine("=====
            =====\n\n===== SERVER STARTED
            =====\n\n=====
            ==\n");

            while (true)
            {
                TcpClient client = listener.AcceptTcpClient();
                client.ReceiveTimeout = timeout;
                counter++;
                Client_unit clientObject = new
                Client_unit(client, counter, max_count);
                clientObject.onDistruct += DecreaseCounter;

                // создаем новый поток для обслуживания
                нового клиента
                Thread clientThread = new Thread(new
                ThreadStart(clientObject.Process));
                clientThread.Start();
            }
        }
        catch (Exception ex) {
            Console.WriteLine(ex.Message); }
        finally
        {
            if (listener != null)
                listener.Stop();
            Console.WriteLine("Для завершения жми
            \nENTER\n");
            Console.ReadLine();
        }
    }
}
```

```
}
}
```

Класс-обработчик клиентов:

```
class Client_unit
{
    private int counter;
    private int max_count;

    public delegate void DecreaseCount();
    public event DecreaseCount onDistruct;

    public TcpClient client;
    NetworkStream stream;

    public Client_unit(TcpClient tcpClient, int counter, int
    max_count)
    {
        client = tcpClient;
        this.counter = counter;
        this.max_count = max_count;
    }

    public void Process()
    {
        DB_Process db = null;
        stream = client.GetStream();
        try
        {
            while (true)
            {
                string message = Get();
                Console.WriteLine($"{(db != null ?
                db.User_name : "-----")}: {message}");

                string code = message.Substring(0, 3);
                string text = message.Substring(5).Trim();
                switch (code)
                {
                    case "CON": // сообщение подключения
                    {
                        if (counter > max_count)
                        {
                            message = "SERVER OVERFLOW .
                            DISCONNECTED";
                            Send(message);
                            throw new Exception(message);
                        }
                    }

                    try
                    {

```

```

        db = new DB_Process
        (
            text.Substring(0, text.IndexOf('~')),
            text.Substring(text.IndexOf('~') + 1)
        );
    }
    catch (Exception ex)
    {
        throw new Exception(text.Substring(0,
text.IndexOf('~')), ex);
    }

    var parent_roles =
db.One_List.FromSqlRaw("SELECT * FROM
get_parent_roles(current_user)").ToList();
    StringBuilder builder = new
StringBuilder("CONNECTED");
    foreach (var role in parent_roles)
builder.Append($"{role != null ? role.Field_1 : "null"}");
    Send(builder.ToString());
}
break;

#region VIEWS

case "C00": // текстовое тестовое
сообщение
{
    Send(text.ToUpper());
}
break;
case "C01":
{
    try
    {
        var list =
db.Two_List.FromSqlRaw("SELECT * FROM
c01_2_fields").ToList();
        StringBuilder builder = new
StringBuilder();
        foreach (var item in list)
builder.Append($"{r}\n~{item.Field_1}~{item.Field_2}");
        message = builder.ToString();
    }
    catch (Exception ex)
    {
        message = $"{ex.Message} .
EXCEPTION";
        Console.WriteLine($"{db.User_name}:
{message}");
    }
    Send(message);
}
break;
case "C02":
{
    try
    {

```

```

        var list =
db.Three_List.FromSqlRaw("SELECT * FROM
c02_3_fields").ToList();
        StringBuilder builder = new
StringBuilder();
        foreach (var item in list)
builder.Append($"{r}\n~{item.Field_1}~{item.Field_2}~{ite
m.Field_3}");
        message = builder.ToString();
    }
    catch (Exception ex)
    {
        message = $"{ex.Message} .
EXCEPTION";
        Console.WriteLine($"{db.User_name}:
{message}");
    }
    Send(message);
}
break;
case "C03":
{
    try
    {
        var list =
db.Two_List.FromSqlRaw("SELECT * FROM
c03_2_fields").ToList();
        StringBuilder builder = new
StringBuilder();
        foreach (var item in list)
builder.Append($"{r}\n~{item.Field_1}~{item.Field_2}");
        message = builder.ToString();
    }
    catch (Exception ex)
    {
        message = $"{ex.Message} .
EXCEPTION";
        Console.WriteLine($"{db.User_name}:
{message}");
    }
    Send(message);
}
break;
case "C04":
{
    try
    {
        var list =
db.Three_List.FromSqlRaw("SELECT * FROM
c04_3_fields").ToList();
        StringBuilder builder = new
StringBuilder();
        foreach (var item in list)
builder.Append($"{r}\n~{item.Field_1}~{item.Field_2}~{ite
m.Field_3}");
        message = builder.ToString();
    }
    catch (Exception ex)
    {

```



```

        message = $"{ex.Message} .
EXCEPTION";
        Console.WriteLine($"{db.User_name}:
{message}");
    }
    Send(message);
}
break;
case "C05":
{
    try
    {
        var list =
db.Two_List.FromSqlRaw("SELECT * FROM
c05_2_fields").ToList();
        StringBuilder builder = new
StringBuilder();
        foreach (var item in list)
builder.Append($"{r\n~{item.Field_1}~{item.Field_2}");
        message = builder.ToString();
    }
    catch (Exception ex)
    {
        message = $"{ex.Message} .
EXCEPTION";
        Console.WriteLine($"{db.User_name}:
{message}");
    }
    Send(message);
}
break;
case "C06":
{
    try
    {
        var list =
db.Three_List.FromSqlRaw("SELECT * FROM
c06_3_fields").ToList();
        StringBuilder builder = new
StringBuilder();
        foreach (var item in list)
builder.Append($"{r\n~{item.Field_1}~{item.Field_2}~{ite
m.Field_3}");
        message = builder.ToString();
    }
    catch (Exception ex)
    {
        message = $"{ex.Message} .
EXCEPTION";
        Console.WriteLine($"{db.User_name}:
{message}");
    }
    Send(message);
}
break;

#endregion VIEWS

#region FUNCTIONS

```

```

        case "C10":
        {
            try
            {
                string[] arguments = text.Split(new[] {
"" }, StringSplitOptions.RemoveEmptyEntries);
                if (arguments.Length < 1) throw new
Exception("not enough arguments");
                var list =
db.Two_List.FromSqlRaw($"SELECT * FROM
c10_2_fields('{arguments[0]}')").ToList();

                StringBuilder builder = new
StringBuilder();
                foreach (var item in list)
builder.Append($"{r\n~{item.Field_1}~{item.Field_2}");
                message = builder.ToString();
            }
            catch (Exception ex)
            {
                message = $"{ex.Message} .
EXCEPTION";
                Console.WriteLine($"{db.User_name}:
{message}");
            }
            Send(message);
        }
        break;
        case "C17":
        {
            try
            {
                string[] arguments = text.Split(new[] {
"" }, StringSplitOptions.RemoveEmptyEntries);
                if (arguments.Length < 1) throw new
Exception("not enough arguments");
                var list =
db.Four_List.FromSqlRaw($"SELECT * FROM
c17_4_fields('{arguments[0]}')").ToList();

                StringBuilder builder = new
StringBuilder();
                foreach (var item in list)
builder.Append($"{r\n~{item.Field_1}~{item.Field_2}~{ite
m.Field_3}~{item.Field_4}");
                message = builder.ToString();
            }
            catch (Exception ex)
            {
                message = $"{ex.Message} .
EXCEPTION";
                Console.WriteLine($"{db.User_name}:
{message}");
            }
            Send(message);
        }
        break;

```

```

case "C18":
{
}
break;
case "C19":
{
}
break;
case "C20":
{
}
break;

#endregion FUNCTIONS

#region OTHER QUERIES

case "C21": // SELECT Clients
{
    try
    {
        var list =
db.Eight_List.FromSqlRaw($"SELECT * FROM
c21_8_fields()").ToList();

        StringBuilder builder = new
StringBuilder();

        foreach (var item in list)

builder.Append($"\\r\\n~{item.Field_1}~{item.Field_2}~{ite
m.Field_3}~{item.Field_4}~{item.Field_5}~{item.Field_6}~{
item.Field_7}~{item.Field_8}");

        message = builder.ToString();
    }
    catch (Exception ex)
    {
        message = $"{ex.Message} .
EXCEPTION";
        Console.WriteLine($"{db.User_name}:
{message}");
    }
    Send(message);
}
break;
case "C31": // INSERT Client
{
    try
    {
        string[] arguments = text.Split(new[] {
"~" }, StringSplitOptions.RemoveEmptyEntries);
        if (arguments.Length < 6) throw new
Exception("not enough arguments");
        var list = db.One_List.FromSqlRaw(
        $"SELECT * FROM
c31_1_field('{arguments[0]}', '{arguments[1]}',
'{arguments[2]}', '{arguments[3]}', '{arguments[4]}',
'{arguments[5]}')");

```

```

        StringBuilder builder = new
StringBuilder();

        foreach (var item in list)
builder.Append($"\\r\\n~{item.Field_1}");
        message = builder.ToString();
    }
    catch (Exception ex)
    {
        message = $"{ex.Message} .
EXCEPTION";
        Console.WriteLine($"{db.User_name}:
{message}");
    }
    Send(message);
}
break;
case "C41": // UPDATE Clients
{
    try
    {
        string[] sides = text.Split(new[] { " |:" },
StringSplitOptions.RemoveEmptyEntries);
        string[] arguments = sides[0].Split(new[]
{ "~" }, StringSplitOptions.None);
        string[] ids = sides[1].Split(new[] { "~" },
StringSplitOptions.RemoveEmptyEntries);

        StringBuilder query = new
StringBuilder("WITH rows AS (UPDATE \"Clients\" SET");
        if (arguments[1].Length > 0)
query.Append($" , surname='{arguments[1]}');
        if (arguments[2].Length > 0)
query.Append($" , name='{arguments[2]}');
        if (arguments[3].Length > 0)
query.Append($" , fathername='{arguments[3]}');
        if (arguments[4].Length > 0)
query.Append($" , birthday='{arguments[4]}');
        if (arguments[5].Length > 0)
query.Append($" , \"group\"=(SELECT id FROM \"Groups\"
WHERE name='{arguments[5]}')");
        if (arguments[6].Length > 0)
query.Append($" , address='{arguments[6]}');
        query.Replace("SET, ", "SET");
        query.Append(" WHERE id IN (");
        foreach (string id in ids)
query.Append($" , {id}");
        query.Replace(",", "(");
        query.Append(") RETURNING 1) SELECT
count(*)::text AS field_1 FROM rows;");

        var list =
db.One_List.FromSqlRaw(query.ToString()).ToList();

        StringBuilder builder = new
StringBuilder();

        foreach (var item in list)

builder.Append($"\\r\\n~{item.Field_1}");
        message = builder.ToString();

```

```

    }
    catch (Exception ex)
    {
        message = $"{ex.Message} .
EXCEPTION";
        Console.WriteLine($"{db.User_name}:
{message}");
    }
    Send(message);
}
break;
case "C51": // DELETE Clients
{
    try
    {
        string[] ids = text.Split(new[] { "~" },
StringSplitOptions.RemoveEmptyEntries);

        StringBuilder query = new
StringBuilder("WITH rows AS (DELETE FROM \"Clients\"
WHERE id IN (");
        foreach (string id in ids)
        query.Append($"", {id}");
        query.Replace(",", "(");
        query.Append(") RETURNING 1) SELECT
count(*)::text AS field_1 FROM rows;");

        var list =
db.One_List.FromSqlRaw(query.ToString()).ToList();

        StringBuilder builder = new
StringBuilder();
        foreach (var item in list)

        builder.Append($"{r\n~{item.Field_1}");
        message = builder.ToString();
    }
    catch (Exception ex)
    {
        message = $"{ex.Message} .
EXCEPTION";
        Console.WriteLine($"{db.User_name}:
{message}");
    }
    Send(message);
}
break;
case "C71": // SELECT Clients [alter]
{
    try
    {
        var list =
db.Two_List.FromSqlRaw($"SELECT * FROM
C71_2_fields(").ToList();

        StringBuilder builder = new
StringBuilder();
        foreach (var item in list)
        {

```

```

        builder.Append($"{r\n~{item.Field_1}~{item.Field_2}");
        message = builder.ToString();
    }
    catch (Exception ex)
    {
        message = $"{ex.Message} .
EXCEPTION";
        Console.WriteLine($"{db.User_name}:
{message}");
    }
    Send(message);
}
break;
case "C61": // SELECT Workers
{
    try
    {
        var list =
db.One_List.FromSqlRaw($"SELECT * FROM
c61_1_fields(").ToList();

        StringBuilder builder = new
StringBuilder();
        foreach (var item in list)

        builder.Append($"{r\n~{item.Field_1}");
        message = builder.ToString();
    }
    catch (Exception ex)
    {
        message = $"{ex.Message} .
EXCEPTION";
        Console.WriteLine($"{db.User_name}:
{message}");
    }
    Send(message);
}
break;
case "C62": // ADD Worker
{
    try
    {
        string[] arguments = text.Split(new[] {
"~" }, StringSplitOptions.RemoveEmptyEntries);
        if (arguments.Length < 2) throw new
Exception("not enough arguments");
        db.Database.ExecuteSqlRaw(
$@"CREATE ROLE {arguments[0]} WITH LOGIN
NOSUPERUSER NOCREATEDB NOCREATEROLE INHERIT
NOREPLICATION
CONNECTION LIMIT 1 PASSWORD '{arguments[1]}';
GRANT worker_gp TO {arguments[0]};
CREATE TABLE journal_{arguments[0]} PARTITION OF
journal FOR VALUES IN ('{arguments[0]}')");

        message = "SUCCESS";
    }

```

```

    }
    catch (Exception ex)
    {
        message = $"{ex.Message} .
EXCEPTION";
        Console.WriteLine($"{db.User_name}:
{message}");
    }
    Send(message);
}
break;
case "C63": // UPDATE Worker
{
    try
    {
        string[] sides = text.Split(new[] { " |:"
" }, StringSplitOptions.RemoveEmptyEntries);
        string[] arguments = sides[0].Split(new[]
{ "~" }, StringSplitOptions.None);
        string login = sides[1].Substring(1);

        StringBuilder query = new
StringBuilder();
        if (arguments[1].Length > 0)
            query.Append($"ALTER ROLE {login} RENAME TO
{arguments[1]}; ALTER TABLE IF EXISTS journal_{login}
RENAME TO journal_{arguments[1]};");
        if (arguments[2].Length > 0)
            query.Append($"ALTER ROLE {login} PASSWORD
'{arguments[2]}';");

        db.Database.ExecuteSqlRaw(query.ToString());
        message = "SUCCESS";
    }
    catch (Exception ex)
    {
        message = $"{ex.Message} .
EXCEPTION";
        Console.WriteLine($"{db.User_name}:
{message}");
    }
    Send(message);
}
break;
case "C64": // DELETE Worker
{
    try
    {
        text = text.Substring(1);
        string query =
$@"DROP ROLE {text};
UPDATE journal SET user_name = user_name || ' [del]'
WHERE user_name = '{text}';
DROP TABLE IF EXISTS journal_{text}";
        var list =
db.Database.ExecuteSqlRaw(query.ToString());
        message = "SUCCESS";
    }

```

```

    catch (Exception ex)
    {
        message = $"{ex.Message} .
EXCEPTION";
        Console.WriteLine($"{db.User_name}:
{message}");
    }
    Send(message);
}
break;

#endregion OTHER QUERIES

case "DIS": // сообщение отключения
{
    throw new Exception("CLOSED");
}
default:
{
    message = $"unknown command .
EXCEPTION";
    Console.WriteLine($"{db.User_name}:
{message}");
    Send(message);
}
break;
}
}
catch (System.IO.IOException ex) when
(ex.Message.Contains("требуемое время"))
{
    string message = "OVERTIME . DISCONNECTED";
    Console.WriteLine($"{db.User_name}:
{message}");
    Send(message);
}
catch (Exception ex)
{
    string message = $"{ex.Message} .
DISCONNECTED";

    if (ex.Message.Contains("CLOSED"))
    {
        Console.WriteLine($"{db.User_name}:
{message}");
    }
    else
    {
        if (ex.InnerException != null)
        {
            message = $"DataBase .
{ex.InnerException.Message} . DISCONNECTED";
            Console.WriteLine($"{ex.Message}:
{message}");
        }
        else Console.WriteLine($"{db.User_name}:
{message}");
    }
}

```

```

        Send(message);
    }
}
finally
{
    onDistruct();

    if (stream != null)
    { stream.Close(); }

    if (client != null)
    { client.Close(); }

    if (db != null)
    { db.Dispose(); db = null; }
}

private string Get()
{
    StringBuilder builder = new StringBuilder(); //
получаем сообщение
    byte[] data = new byte[64];           // буфер для
данных
    int bytes = 0;

```

```

        do
        {
            bytes = stream.Read(data, 0, data.Length);

            builder.Append(Encoding.Unicode.GetString(data, 0,
bytes));
        }
        while (stream.DataAvailable);

        builder.Replace("", "");
        return builder.ToString();
    }

    private void Send(string message)
    {
        if (message.Length == 0) message = "nothing";
        byte[] data =
Encoding.Unicode.GetBytes(message);
        stream.Write(data, 0, data.Length);
    }
}

```

ПРИЛОЖЕНИЕ В.

Листинг клиентского приложения

Основной класс клиента:

```

public partial class FORM_main : Form
{
    private FORM_authorization auth;
    private FORM_diagram_AND_excel diagram;
    private FORM_t_group groups;
    private Form menu;

    private const int port = 8888;
    private const string address = "127.0.0.1";

    private static TcpClient client;
    private static NetworkStream stream;
    #if DEBUG
        private bool postgres = false;
    #endif
    private bool director = false;
    private bool manager = false;
    private bool worker = false;

    string[] rows;

    #region T_MENU ITEMS

    ToolStripMenuItem ins_client;
    ToolStripMenuItem upd_client;
    ToolStripMenuItem del_client;

    ToolStripMenuItem ins_barber;
    ToolStripMenuItem upd_barber;
    ToolStripMenuItem del_barber;

    ToolStripMenuItem ins_journal;
    ToolStripMenuItem upd_journal;
    ToolStripMenuItem del_journal;

    ToolStripMenuItem add_worker;
    ToolStripMenuItem upd_worker;
    ToolStripMenuItem del_worker;

    ToolStripMenuItem show_diagram;

    #endregion T_MENU ITEMS

    #region FORM CONTROL

    public FORM_main(FORM_authorization auth_form)
    {
        InitializeComponent();

        ins_client = new ToolStripMenuItem("Добавить
клиента");

        ins_client.Click += ins_client_Click;
        upd_client = new ToolStripMenuItem("Изменить
выделенных клиентов");
        upd_client.Click += upd_client_Click;
        del_client = new ToolStripMenuItem("Удалить
выделенных клиентов");
        del_client.Click += del_client_Click;

        ins_barber = new ToolStripMenuItem("Добавить
парикмахерскую");
        ins_barber.Click += ins_barber_Click;
        upd_barber = new ToolStripMenuItem("Изменить
выделенные парикмахерские");
        upd_barber.Click += upd_barber_Click;
        del_barber = new ToolStripMenuItem("Удалить
выделенные парикмахерские");
        del_barber.Click += del_barber_Click;

        ins_journal = new ToolStripMenuItem("Добавить
заказ");
        ins_journal.Click += ins_journal_Click;
        upd_journal = new ToolStripMenuItem("Изменить
выделенные заказы");
        upd_journal.Click += upd_journal_Click;
        del_journal = new ToolStripMenuItem("Удалить
выделенные заказы");
        del_journal.Click += del_journal_Click;

        add_worker = new ToolStripMenuItem("Создать
аккаунт работника");
        add_worker.Click += add_worker_Click;
        upd_worker = new ToolStripMenuItem("Изменить
выделенный аккаунт работника");
        upd_worker.Click += upd_worker_Click;
        del_worker = new ToolStripMenuItem("Удалить
выделенный аккаунт работника");
        del_worker.Click += del_worker_Click;

        show_diagram = new
ToolStripMenuItem("Показать в виде диаграммы");
        show_diagram.Click += show_diagram_Click;

        // OTHER FORMS
        auth = auth_form;

        groups = new FORM_t_group(this);
        ToolStripMenuItem ins_group = new
ToolStripMenuItem("Добавить клиентскую группу");
        ins_group.Click += ins_group_Click;
        ToolStripMenuItem del_group = new
ToolStripMenuItem("Удалить клиентскую группу");
        del_group.Click += del_group_Click;
    }
}

```

```

        groups.CONTEXT_menu.Items.AddRange(new[] {
            ins_group, del_group });
    }

    private void FORM_main_FormClosed(object sender,
        FormClosedEventArgs e)
    {
        if (Visible)
        {
            string message = "DIS: CONNECT";
            byte[] data =
                Encoding.Unicode.GetBytes(message);
            stream.Write(data, 0, data.Length);

            client.Close();
            Application.Exit();
        }
    }

    private void FORM_main_VisibleChanged(object
        sender, EventArgs e)
    {
        if (menu != null && !menu.IsDisposed)
            menu.Visible = Visible;
    }

    private void BTN_menu_Click(object sender,
        EventArgs e)
    {
        if (menu != null && menu.Visible) menu.Close();
        else
        {
            if (menu == null || menu.IsDisposed)
            {
                #if DEBUG
                    if (postgres) menu = new FORM_menu_p(this);
                #endif
                if (director) menu = new FORM_menu_d(this);
                if (manager) menu = new
                    FORM_menu_m(this);
                if (worker) menu = new FORM_menu_w(this);
            }
            menu.Show();
        }
    }

    private void Show_page()
    {
        for (int i = 0; i < DGRID_table.Rows.Count; i++)
        {
            if (i >= (NUM_page.Value - 1) * 17 && i <
                NUM_page.Value * 17) DGRID_table.Rows[i].Visible =
                true;
            else DGRID_table.Rows[i].Visible = false;
        }
    }

    private void NUM_page_ValueChanged(object
        sender, EventArgs e)
    {

```

```

        Show_page();
    }

    private void NUM_page_KeyDown(object sender,
        KeyEventArgs e)
    {
        if (e.KeyCode == Keys.Enter)
            { e.SuppressKeyPress = true; }
    }

    private void TXT_search_Leave(object sender,
        EventArgs e)
    {
        if (TXT_search.Text == "") { SetRows(rows);
            NUM_page.Value = 1; Show_page(); }
    }

    private void TXT_search_KeyDown(object sender,
        KeyEventArgs e)
    {
        if (e.KeyCode == Keys.Enter)
        {
            e.SuppressKeyPress = true;
            if (TXT_search.Text != "")
            {
                SetRows(rows.Where(line =>
                    line.Contains(TXT_search.Text.Trim())).ToArray());
            }
            else
            {
                SetRows(rows);
            }
            NUM_page.Value = 1;
            Show_page();
        }
        if (e.KeyCode == Keys.Escape)
        {
            e.SuppressKeyPress = true;
            SelectNextControl((Control)sender, true, true,
                true, true);
        }
    }

    private void BTN_search_Click(object sender,
        EventArgs e)
    {
        NUM_page.Value = 1; SetRows(rows.Where(line =>
            line.Contains(TXT_search.Text.Trim())).ToArray());
        Show_page();
    }

    private void BTN_clear_Click(object sender,
        EventArgs e)
    {
        NUM_page.Value = 1; TXT_search.Text = "";
        SetRows(rows); Show_page();
    }

    #endregion FORM CONTROL

    private string Send(string message)
    {
        StringBuilder builder = new StringBuilder();

```

```

        byte[] data =
Encoding.Unicode.GetBytes(message); // преобразуем
сообщение в массив байтов
        try
        {
            stream.Write(data, 0, data.Length);    //
отправка сообщения

            // получаем ответ
            data = new byte[64];                // буфер для
получаемых данных
            int bytes = 0;
            do
            {
                bytes = stream.Read(data, 0, data.Length);

builder.Append(Encoding.Unicode.GetString(data, 0,
bytes));
            }
            while (stream.DataAvailable);
        }
        catch
        {
            builder.Append("OVERTIME . DISCONNECTED");
        }

        return builder.ToString();
    }

    public bool CON_query(string user_name, string
password)
    {
        try
        {
            client = new TcpClient(address, port);
            stream = client.GetStream();

            string message = $"CON:
{user_name}~{password}";
            message = Send(message);
            if (message.Contains("DISCONNECTED")) {
client.Close(); return false; }

            if (user_name == "postgres")
            {
#if DEBUG
                postgres = true;
#else
                message = "DIS: CONNECT --- MISHANDLING";
                byte[] data =
Encoding.Unicode.GetBytes(message);
                stream.Write(data, 0, data.Length);
                client.Close();
                return false;
#endif
            }
        }
        else if (message.Contains("director_gp"))
        {
            director = true;

```

```

        }
        else if (message.Contains("manager_gp"))
        {
            manager = true;
        }
        else if (message.Contains("worker_gp"))
        {
            worker = true;
        }
    }
    catch (Exception ex)
    {
        if (ex.Message.Contains("отверг"))
        MessageBox.Show("Сервер: OFFLINE");
        else
        MessageBox.Show(ex.Message);

        if (client != null) client.Close();
        return false;
    }

    auth.Hide();
    Show();

    return true;
}

public void DIS_query()
{
    string message = "DIS: CONNECT";
    byte[] data =
Encoding.Unicode.GetBytes(message);
    stream.Write(data, 0, data.Length);

    if (client != null) client.Close();

    auth.Show();
    Hide();

    TXT_title.Text = "";
    DGRID_table.Columns.Clear();
    DGRID_table.Rows.Clear();
    groups.Hide();
    menu.Close();

#if DEBUG
    postgres = false;
#endif
    director = false; manager = false; worker = false;
}

#region VIEWS

/// <summary>
/// Count of Barbers by it's factors
/// </summary>
public void C01_query()
{

```



```

        Query("C01", "Count of Barbers by it's factors",
new[] { new KeyValuePair<string, string>("count", "count
of Barbers"),

                                new
KeyValuePair<string, string>("factor", "factor for costs"))});
    }

    /// <summary>
    /// Barbers with it's ownerships & ranks
    /// </summary>
    public void C02_query()
    {
        Query("C02", "Barbers with it's ownerships &
ranks", new[] { new KeyValuePair<string, string>("barber",
"Barbershop"),

                                new
KeyValuePair<string, string>("ownership", "Ownership"),

                                new
KeyValuePair<string, string>("rank", "Rank"))});
    }

    /// <summary>
    /// Count of barbers at districts
    /// </summary>
    public void C03_query()
    {
        Query("C03", "Count of barbers at districts", new[]
{ new KeyValuePair<string, string>("count", "count of
Barbers"),

                                new
KeyValuePair<string, string>("district", "district"))});
    }

    /// <summary>
    /// Unranked Barbers at districts
    /// </summary>
    public void C04_query()
    {
        Query("C04", "Unranked Barbers at districts",
new[] { new KeyValuePair<string, string>("barber ",
"Barber"),

                                new
KeyValuePair<string, string>("date", "Starting date"),

                                new
KeyValuePair<string, string>("district", "District"))});
    }

    /// <summary>
    /// Count of clients in groups
    /// </summary>
    public void C05_query()
    {
        Query("C05", "Count of clients in groups", new[] {
new KeyValuePair<string, string>("count", "count of
Clients"),

                                new
KeyValuePair<string, string>("group", "Group"))});

```

```

        CONTEXT_menu.Items.AddRange(new[] {
show_diagram });
    }

    /// <summary>
    /// Clients in groups
    /// </summary>
    public void C06_query()
    {
        Query("C06", "Clients in groups", new[] { new
KeyValuePair<string, string>("surname", "surname"),

                                new KeyValuePair<string,
string>("name", "name"),

                                new KeyValuePair<string,
string>("group", "group"))});
    }

    /// <summary>
    /// Clients orders
    /// </summary>
    public void C07_query()
    {
        Query("C07", "Clients orders", new[] { new
KeyValuePair<string, string>("surname", "surname"),

                                new KeyValuePair<string,
string>("barber", "Barber"),

                                new KeyValuePair<string,
string>("date", "record date"))});
    }

    /// <summary>
    /// Top profit services
    /// </summary>
    public void C08_query()
    {
        Query("C08", "Top profit services", new[] { new
KeyValuePair<string, string>("count", "count of Records"),

                                new KeyValuePair<string,
string>("profit", "profit"),

                                new KeyValuePair<string,
string>("service", "Service"))});
    }

    /// <summary>
    /// Districts with Barbers
    /// </summary>
    public void C09_query()
    {
        Query("C09", "Districts with Barbers", new[] { new
KeyValuePair<string, string>("district", "District") });
    }

#endregion VIEWS

#region FUNCTIONS

    /// <summary>
    /// Average service order cost
    /// </summary>

```

```

public void C10_query()
{
    Query(true, "C10", "Average service order cost",
        new[] { new KeyValuePair<string, string>("avg",
"average cost"),
        new KeyValuePair<string, string>("service",
"Service")},
        hints: new[] { "service name" });
}

/// <summary>
/// Barbers at ownership
/// </summary>
public void C11_query()
{
    Query(true, "C11", "Barbers at ownership",
        new[] { new KeyValuePair<string,
string>("barber", "Barber"),
        new KeyValuePair<string, string>("ownership",
"Ownership"),
        new KeyValuePair<string, string>("date",
"starting date"),
        new KeyValuePair<string, string>("district",
"District")},
        hints: new[] { "ownership name" });
}

/// <summary>
/// Barbers phones at rank
/// </summary>
public void C12_query()
{
    Query(true, "C12", "Barbers phones at rank",
        new[] { new KeyValuePair<string,
string>("barber", "Barbershop"),
        new KeyValuePair<string, string>("phone",
"phone")},
        hints: new[] { "rank name" });
}

/// <summary>
/// Clients at birthday
/// </summary>
public void C13_query()
{
    Query(true, "C13", "Clients at birthday",
        new[] { new KeyValuePair<string,
string>("group", "Group"),
        new KeyValuePair<string, string>("surname",
"surname"),
        new KeyValuePair<string, string>("birthday",
"birthday")},
        hints: new[] { "birthday [dd.mm.yyyy]" });
}

/// <summary>
/// Clients at groups
/// </summary>
public void C14_query()

```

```

{
    Query(true, "C14", "Clients at groups",
        new[] { new KeyValuePair<string,
string>("surname ", "surname"),
        new KeyValuePair<string, string>("address",
"address")},
        hints: new[] { "group name" });
}

/// <summary>
/// Orders at date
/// </summary>
public void C15_query()
{
    Query(true, "C15", "Orders at date",
        new[] { new KeyValuePair<string,
string>("barber", "Barber"),
        new KeyValuePair<string, string>("cost", "cost"),
        new KeyValuePair<string, string>("date",
"date")},
        hints: new[] { "date [dd.mm.yyyy]" });
}

/// <summary>
/// Profit by expensive services at date
/// </summary>
public void C16_query()
{
    Query(true, "C16", "Profit by expensive services at
date",
        new[] { new KeyValuePair<string,
string>("date", "date"),
        new KeyValuePair<string, string>("sum", "sum
of cost")},
        hints: new[] { "date [dd.mm.yyyy]" });
}

/// <summary>
/// Barbers at rank & factors of other
/// </summary>
public void C17_query()
{
    Query(true, "C17", "Barbers at rank & factors of
other",
        new[] { new KeyValuePair<string, string>("rank",
"Rank"),
        new KeyValuePair<string, string>("factor",
"factor"),
        new KeyValuePair<string, string>("phone",
"phone"),
        new KeyValuePair<string, string>("date",
"starting date")},
        hints: new[] { "rank name" });
}

```

#endregion FUNCTIONS

#region OTHER QUERIES

```

    /// <summary>
    /// Showing CLIENTs & cGROUPs
    /// </summary>
    public void C21_query()
    {
        Query("C21", "Clients", new[] { new
        KeyValuePair<string, string>("id", "id"),
            new KeyValuePair<string,
            string>("surname", "surname"),
            new KeyValuePair<string,
            string>("name", "name"),
            new KeyValuePair<string,
            string>("fathername", "fathername"),
            new KeyValuePair<string,
            string>("birthday", "birthday"),
            new KeyValuePair<string,
            string>("group", "group"),
            new KeyValuePair<string,
            string>("address", "address"),
            new KeyValuePair<string,
            string>("user", "user name")}, new[] { 1, 8 });

        CONTEXT_menu.Items.AddRange(new[] {
        ins_client, upd_client, del_client });
    }

    /// <summary>
    /// Showing BARBERS
    /// </summary>
    public void C22_query()
    {
        Query("C22", "Barbershops", new[] { new
        KeyValuePair<string, string>("id", "id"),
            new KeyValuePair<string,
            string>("district", "district"),
            new KeyValuePair<string,
            string>("rank", "rank"),
            new KeyValuePair<string,
            string>("ownership", "ownership"),
            new KeyValuePair<string,
            string>("date", "starting date"),
            new KeyValuePair<string,
            string>("phone", "phone"),
            new KeyValuePair<string,
            string>("user", "user name")}, new[] { 1, 7 });

        CONTEXT_menu.Items.AddRange(new[] {
        ins_barber, upd_barber, del_barber });
    }

    /// <summary>
    /// Showing JOURNAL
    /// </summary>
    public void C23_query()
    {
        Query("C23", "Journal", new[] { new
        KeyValuePair<string, string>("id", "id"),
            new KeyValuePair<string,
            string>("client", "client"),

```

```

            new KeyValuePair<string,
            string>("barber", "barbershop"),
            new KeyValuePair<string,
            string>("service", "service"),
            new KeyValuePair<string,
            string>("cost", "cost"),
            new KeyValuePair<string,
            string>("date", "date"),
            new KeyValuePair<string,
            string>("user", "user name")}, new[] { 1, 7 });

        CONTEXT_menu.Items.AddRange(new[] {
        ins_journal, upd_journal, del_journal });
    }

    /// <summary>
    /// SELECT workers
    /// </summary>
    public void C61_query()
    {
        Query("C61", "Workers", new[] { new
        KeyValuePair<string, string>("login", "login") });

        CONTEXT_menu.Items.AddRange(new[] {
        add_worker, upd_worker, del_worker });
    }

    /// <summary>
    /// ADD worker
    /// </summary>
    public void C62_query()
    {
        string arguments = InputBox.ShowDialog("Add new
        worker", 2, new[] { "login", "password" });
        if (arguments == null) return;

        string message = Send($"C62: {arguments}");

        if (message.Contains("DISCONNECTED"))
        {
            MessageBox.Show("=== database exception
            ===", "DISCONNECTED");
            client.Close();
            Hide();
            auth.Show();
            return;
        }
        if (message.Contains("EXCEPTION"))
        {
            MessageBox.Show($"=== {message} ===",
            "DATABASE EXCEPTION");
            return;
        }

        MessageBox.Show(message);
        C61_query();
    }

    /// <summary>

```

```

/// UPDATE worker
/// </summary>
public void C63_query()
{
    string user = "";
    foreach (DataGridViewRow row in
DGRID_table.SelectedRows)
    {
        user = row.Cells[0].Value.ToString();
    }
    if (user.Length == 0)
    {
        MessageBox.Show("Need to select worker
before");
        return;
    }

    string arguments = InputBox.ShowDialog("Change
worker", 2, new[] { "login", "password" });
    if (arguments == null) return;

    string message = Send($"C63: {arguments} |:|
{user}");

    if (message.Contains("DISCONNECTED"))
    {
        MessageBox.Show("=== database exception
===", "DISCONNECTED");
        client.Close();
        Hide();
        auth.Show();
        return;
    }
    if (message.Contains("EXCEPTION"))
    {
        MessageBox.Show($"=== {message} ===",
"DATABASE EXCEPTION");
        return;
    }

    MessageBox.Show(message);
    C61_query();
}

/// <summary>
/// DELETE worker
/// </summary>
public void C64_query()
{
    string user = "";
    foreach (DataGridViewRow row in
DGRID_table.SelectedRows)
    {
        user = row.Cells[0].Value.ToString();
    }
    if (user.Length == 0)
    {
        MessageBox.Show("Need to select worker
before");

```

```

        return;
    }

    if (MessageBox.Show($"Are you sure you want\nto
Delete worker \"{user}\"", "Deleting",
MessageBoxButtons.OKCancel) == DialogResult.OK)
    {
        string message = Send($"C64: {user}");

        if (message.Contains("DISCONNECTED"))
        {
            MessageBox.Show("=== database exception
===", "DISCONNECTED");
            client.Close();
            Hide();
            auth.Show();
            return;
        }
        if (message.Contains("EXCEPTION"))
        {
            MessageBox.Show($"=== {message} ===",
"DATABASE EXCEPTION");
            return;
        }

        MessageBox.Show(message);
        C61_query();
    }
}

#endregion OTHER QUERIES

#region T_MENU ITEMS PROC

void ins_group_Click(object sender, EventArgs e)
{
    Insert("C34", "New group", hints: new[] { "name"
});

    Groups();
}

void del_group_Click(object sender, EventArgs e)
{
    StringBuilder rows = new StringBuilder();
    foreach (DataGridViewRow row in
groups.DGRID_table.SelectedRows)
    {
        rows.Append($"~{row.Cells[0].Value}");
    }
    if (rows.Length == 0)
    {
        MessageBox.Show("Need to select rows
before");
        return;
    }

    if (MessageBox.Show($"Are you sure you want\nto
Delete {groups.DGRID_table.SelectedRows.Count} row(s)

```

```

and related", "Deleting", MessageBoxButtons.OKCancel)
== DialogResult.OK)
    {
        Process($"C54: {rows.ToString()}");
    }

    C21_query();
    Groups();
}

void del_worker_Click(object sender, EventArgs e)
{
    C64_query();
}

void show_diagram_Click(object sender, EventArgs e)
{
    if (diagram != null && !diagram.IsDisposed)
diagram.Close();
    else
    {
        diagram = new FORM_diagram_AND_excel(this);
        diagram.Show();
    }
}

#endregion T_MENU ITEMS PROC

private void Query(string code, string title,
KeyValuePair<string, string>[] columns, int[] hidden = null)
{
    CONTEXT_menu.Items.Clear();

    string message = $" {code}: ---";
    message = Send(message);
    if (message.Contains("DISCONNECTED"))
    {
        MessageBox.Show("=== database exception
===, "DISCONNECTED");
        client.Close();
        Hide();
        auth.Show();
        return;
    }

    DGRID_table.Columns.Clear();
    DGRID_table.Rows.Clear();

    if (message.Contains("EXCEPTION"))
    {
        MessageBox.Show($"=== {message} ===",
"DATABASE EXCEPTION");
        return;
    }

    TXT_title.Text = title;

    for (int i = 0; i < columns.Length; i++)

```

```

    {
        DGRID_table.Columns.Add(columns[i].Key,
columns[i].Value);
        if (hidden != null && hidden.Contains(i + 1))
DGRID_table.Columns[i].Visible = false;
    }

    if (code != "C61") DGRID_table.MultiSelect = true;
    else DGRID_table.MultiSelect = false;

    rows = message.Split(new[] { "\r\n" },
StringSplitOptions.RemoveEmptyEntries);
    if (code != "C22") SetRows(rows);
    else SetRows(rows.Where(line =>
!line.Contains("upd") && !line.Contains("del")).ToArray());

    Show_page();
    if (code != "C21") groups.Hide();
    else Groups();
}

private void Query(bool needSelection, string code,
string title,
KeyValuePair<string, string>[] columns,
int[] hidden = null,
short input_fields = 1, string[] hints = null)
{
    CONTEXT_menu.Items.Clear();

    string arguments = InputBox.ShowDialog(title,
input_fields, hints);
    if (arguments == null) return;

    string message = $" {code}: {arguments}";
    message = Send(message);
    if (message.Contains("DISCONNECTED"))
    {
        MessageBox.Show("=== database exception
===, "DISCONNECTED");
        client.Close();
        Hide();
        auth.Show();
        return;
    }

    DGRID_table.Columns.Clear();
    DGRID_table.Rows.Clear();

    if (message.Contains("EXCEPTION"))
    {
        MessageBox.Show($"=== {message} ===",
"DATABASE EXCEPTION");
        return;
    }

    TXT_title.Text = title;

    for (int i = 0; i < columns.Length; i++)
    {

```

```

        DGRID_table.Columns.Add(columns[i].Key,
columns[i].Value);
        if (hidden != null && hidden.Contains(i + 1))
DGRID_table.Columns[i].Visible = false;
    }

    DGRID_table.MultiSelect = true;

    rows = message.Split(new[] { "\r\n" },
StringSplitOptions.RemoveEmptyEntries);
    SetRows(rows);

    Show_page();
    groups.Hide();
}

private void SetRows(string[] rows)
{
    DGRID_table.Rows.Clear();
    foreach (string row in rows)
    {
        string[] items = row.Split(new[] { "~" },
StringSplitOptions.RemoveEmptyEntries);
        DGRID_table.Rows.Add(items);
    }

    NUM_page.Value = 1;
    NUM_page.Maximum = rows.Length / 17;
    if ((rows.Length % 17) > 0)
NUM_page.Maximum++;
    TXT_max.Text = $"\\ {NUM_page.Maximum}";
}

private void Groups()
{
    string message = "C24: ---";
    message = Send(message);
    if (message.Contains("DISCONNECTED"))
    {
        MessageBox.Show("=== database exception
===", "DISCONNECTED");
        client.Close();
        Hide();
        auth.Show();
        return;
    }

    groups.DGRID_table.Columns.Clear();
    groups.DGRID_table.Rows.Clear();

    if (message.Contains("EXCEPTION"))
    {
        MessageBox.Show($"=== {message} ===",
"DATABASE EXCEPTION");
        return;
    }

    groups.DGRID_table.Columns.Add("id", "id");

```

```

        groups.DGRID_table.Columns.Add("group",
"group");
        groups.DGRID_table.Columns[0].Visible = false;

        string[] rows = message.Split(new[] { "\r\n" },
StringSplitOptions.RemoveEmptyEntries);
        foreach (string row in rows)
        {
            string[] items = row.Split(new[] { "~" },
StringSplitOptions.RemoveEmptyEntries);
            groups.DGRID_table.Rows.Add(items);
        }

        groups.Show();
    }

    private List<KeyValuePair<string, string>>
Get_DB_list(string code)
    {
        string message = $" {code}: ---";
        message = Send(message);
        if (message.Contains("DISCONNECTED"))
        {
            MessageBox.Show("=== database exception
===", "DISCONNECTED");
            client.Close();
            Hide();
            auth.Show();
            return null;
        }

        if (message.Contains("EXCEPTION"))
        {
            MessageBox.Show($"=== {message} ===",
"DATABASE EXCEPTION");
            return null;
        }

        string[] rows = message.Split(new[] { "\r\n" },
StringSplitOptions.RemoveEmptyEntries);
        List<KeyValuePair<string, string>> result = new
List<KeyValuePair<string, string>>();
        foreach (string row in rows)
        {
            string[] items = row.Split(new[] { "~" },
StringSplitOptions.RemoveEmptyEntries);
            result.Add(new KeyValuePair<string,
string>(items[0], items[1]));
        }

        return result;
    }

    private void Process(string message)
    {
        message = Send(message);

        if (message.Contains("DISCONNECTED"))
        {

```

```

        MessageBox.Show("=== database exception
===", "DISCONNECTED");
        client.Close();
        Hide();
        auth.Show();
        return;
    }
    if (message.Contains("EXCEPTION"))
    {
        MessageBox.Show($"=== {message} ===",
"DATABASE EXCEPTION");
        return;
    }

    MessageBox.Show($" {message.Substring(3)} row(s)
affected");
    }

    private void Insert(string code, string title, short
input_fields = 1, string[] hints = null)
    {
        string arguments = InputBox.ShowDialog(title,
input_fields, hints);
        if (arguments == null) return;

        Process($"{code}: {arguments}");
    }

    private void Update(string code, string title, short
input_fields = 1, string[] hints = null)
    {
        StringBuilder rows = new StringBuilder();
        foreach (DataGridViewRow row in
DGRID_table.SelectedRows)
        {
            rows.Append($"{row.Cells[0].Value}");
        }
        if (rows.Length == 0)
        {

```

```

        MessageBox.Show("Need to select rows
before");
        return;
    }

    string arguments = InputBox.ShowDialog(title,
input_fields, hints);
    if (arguments == null) return;

    Process($"{code}: {arguments} | :|
{rows.ToString()}");
    }

    private void Delete(string code)
    {
        StringBuilder rows = new StringBuilder();
        foreach (DataGridViewRow row in
DGRID_table.SelectedRows)
        {
            rows.Append($"{row.Cells[0].Value}");
        }
        if (rows.Length == 0)
        {
            MessageBox.Show("Need to select rows
before");
            return;
        }

        if (MessageBox.Show($"Are you sure you want\nto
Delete {DGRID_table.SelectedRows.Count} row(s) and
related", "Deleting", MessageBoxButtons.OKCancel) ==
DialogResult.OK)
        {
            Process($"{code}: {rows.ToString()}");
        }
    }

```

ПРИЛОЖЕНИЕ Г.

Руководство пользователя

Информационная система «Парикмахерские» предназначена для управления данными о работе большого количества различных парикмахерских, расположенных в пределах одного города. Основной функционал системы – предоставление возможности хранить и обрабатывать данные о парикмахерских, клиентах заведений и заказываемых ими услугах.

Система рассчитана на использование сотрудниками и руководителями парикмахерских. Предоставляет возможность директорам заведений просматривать информацию о парикмахерских и вести учёт прибыли, работникам заведений – вести учёт заказов, контролировать регистрацию клиентов, а также свои личные доходы, менеджерам заведений – контролировать списки и успеваемость работников.

Для выполнения описанных функций система снабжена необходимыми областями вывода данных и управляющими меню, открываемыми при нажатии соответствующей кнопки на главном окне клиентского приложения.

ПРИЛОЖЕНИЕ Д.

Руководство администратора

Меню администратора не доступно в стандартном выпуске данной программной системы. Для получения расширенной версии клиентского приложения следует обратиться к разработчикам программного обеспечения.

Панель меню администратора содержит все функции, доступные по отдельности различным пользователям, при этом администраторские права позволяют в полной мере применять данные возможности.

Также администратор имеет полный доступ к базе данных и может просматривать, добавлять и редактировать существующие функции, таблицы, триггеры, поэтому возможность входа как администратора требует ограничений в распространении и должна строго контролироваться.