Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Теоретическая информатика и компьютерные технологии

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ ПО КУРСУ БАЗЫ ДАННЫХ НА ТЕМУ:

База данных стоматологической клиники

Студент _____ Ионов Т.Р.

подпись, дата *фамилия, и.о.*

Научный руководитель _____ Вишняков И.Э.

подпись, дата *фамилия, и.о.*

2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сфера здравоохранения является ключевой в любом современном обществе. Вследствие противоестественного образа жизни, человек встречается с разного рода проблемами со здоровьем. В частности, мягкая пища с большим содержанием сахара приводит к развитию осложнений в полости рта. Оказание качественной и квалифицированной медицинской помощи способно значительно улучшить качество жизни.

Государственные учреждения не способны обеспечить все потребности пациентов. Частные клиники могут предоставить индивидуальное отношение, гибкий график и более открыты к инновациям. Частный сектор, помимо всего прочего, предоставляет возможности карьерного роста и создание личного бренда врача.

В медицинской сфере существует колоссальное количество детальных данных, которые необходимо хранить, обрабатывать и предоставлять с разным уровнем систематизации. С исследовательской точки зрения, это позволяет определить тренд в причинах визита, улучшения или ухудшения здоровья в определенных слоях населения. С точки бизнес-логики, анализ данных позволяет определить выгодные маршруты развития, прозрачность денежного оборота и понятное представление бизнес-процессов.

Целью данной работы является разработка базы данных стоматологической клиники для хранения сотрудников с их рабочим расписанием, клиентов вместе с медицинскими данными, приемов врача и каждой проведенной процедуры и приложения, предоставляющего доступ к данным через понятный интерфейс.

С учетом бизнес-составляющей, требуется разработать приложение, которое принимает информацию о посещении пациентов врача, а также списка процедур, которые были проведены. С учетом бизнес-составляющей – требуется разработать понятный интерфейс для расчета заработной платы сотрудника,

основываясь на его специализации, расписании и сумме от проведенных приемов.

Для выполнения задач необходимо:

- Разработать модель “сущность-связь”;
- Разработать и реализовать реляционную базу данных;
- Создать оконное приложения для ввода и вывода данных;
- Провести тесты системы.

1. Изучение предметной области

Частная медицинская клиника, в целом, состоит из нескольких кабинетов приема, состава врачей разных специальностей, медсестер, администратора.

Клиенты медицинской клиники представляются и как личность, и как обезличенная медицинская карта, храня все необходимые сведения о здоровье.

Управлять частным бизнесом – непростая задача, особенно, если речь идет о человеческом здоровье. Одна из ключевых задач бизнеса – автоматизация процессов. Помимо этого, для сотрудников важна прозрачность процессов формирования заработной платы, поэтому необходимо хранить и уметь предоставлять информацию в понятном виде.

В представлении пациента достаточно хранить данные о его ФИО и номере телефона, все медицинские сведения должны хранится обезличено и независимо.

Поскольку сотрудники клиники работают по сменам, необходимо удобное внесение информации в формате даты и временных промежутков.

Помимо смены, один врач принимает одного пациента в указанном кабинете в указанное время. При необходимости, выписывается рецепт. Посещение включает в себя набор процедур, которые характеризуются кодом в прейскуранте и кодом зуба, над котором процедура была проведена.

2. Разработка базы данных и приложения

2.1 Разработка модели “сущность-связь”

Модель “сущность-связь” – модель данных, позволяющая описывать концептуальные схемы предметной области. На основе требований и ограничений к базе данных, была создана модель сущность-связь (Рисунок 1).

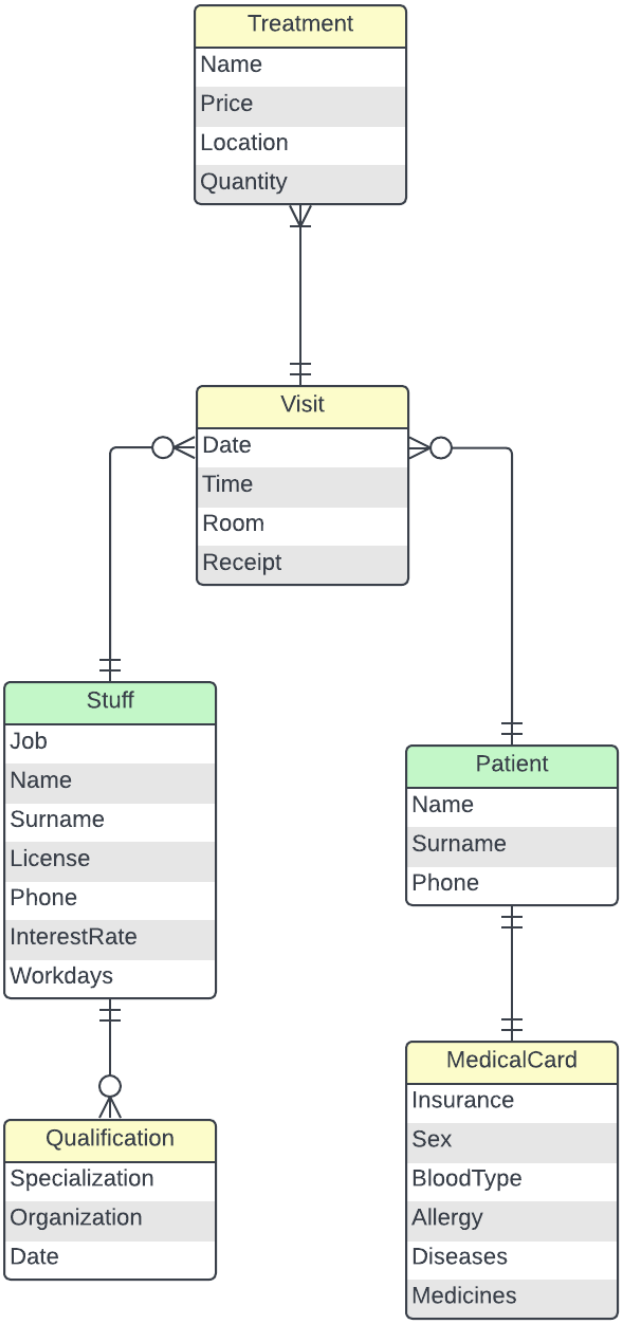

Рисунок 1 - модель сущность-связь

2.2 Разработка реляционной модели

На основе данной модели “сущность-связь” необходимо разработать реляционную модель с помощью этапов:

1. Создание таблицы для каждой сущности:
   1.1. определение первичного ключа (возможно, суррогатного);
   1.2. определение ключей кандидатов;
   1.3. определение свойств каждого столбца:
      1.3.1. тип данных;
      1.3.2. возможность неопределенного значения;
      1.3.3. значение по умолчанию;
      1.3.4. ограничений на значения.
   1.4. проверка нормализации.
2. Создание связей с помощью внешних ключей:
   2.1. Между сильными сущностями (1:1, 1:N, N:M);
   2.2. Для идентификационно-зависимых сущностей;
   2.3. Для слабых сущностей;
   2.4. Для сущностей тип-подтип.
3. Обеспечение условий минимальной кардинальности.

После итераций проектирования, получается реляционная модель, представленная на рисунке 2.

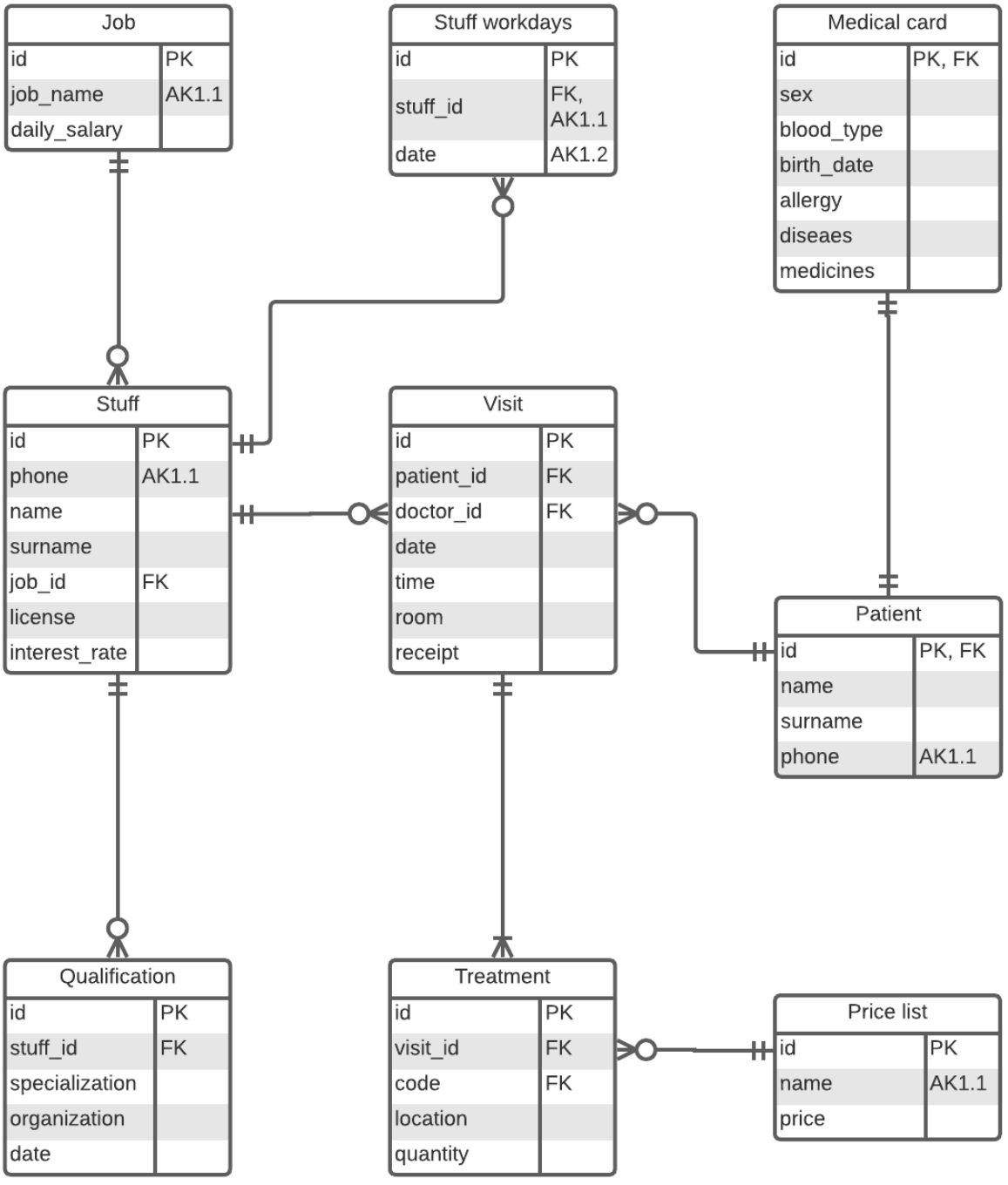

Рисунок 2 - реляционная модель

В ходе преобразований, нормализации и осуществления связей получились 9 сущностей:

1. Stuff – сотрудник клиники.

   Атрибуты:

- phone – номер телефона;
- name – имя сотрудника;
- surname – фамилия сотрудника;
- job_id – идентификатор должности;
- license – лицензия медицинского работника;
- interest_rate – процент за выполненную операцию;

2. Qualification – квалификация сотрудника (врача).

   Атрибуты:

   - stuff_id – идентификатор сотрудника;
   - specialization - код специализации;
   - organization – организация, выдавшая диплом;
   - date - дата вручения диплома.

3. Job – профессия.

   Атрибуты:

   - job_name – наименование должности;
   - daily_salary – плата за смену.

4. Stuff workdays – рабочее расписание.

   Атрибуты:

   - stuff_id – идентификатор сотрудника;
   - date – рабочий день.

5. Patient – клиент клиники.

   Атрибуты:

   - name – имя клиента;
   - surname – фамилия клиента;
   - phone – номер телефона клиента.

6. Medical card – медицинская карта пациента клиники;

   Атрибуты:

   - sex – пол;
   - blood_type – группа крови;

- birth_date – дата рождения;
- allergy – аллергия на препараты;
- diseases – заболевания;
- medicines – принимаемые препараты.

7. Visit – посещение врача клиентом.

   Атрибуты:

   - patient_id – идентификатор пациента;
   - doctor_id – идентификатор врача;
   - date - дата приема;
   - time – время начала приема;
   - room - кабинет приема;
   - receipt – рецепт.

8. Treatment

   Атрибуты:

   - visit_id – идентификатор приема;
   - code – код операции;
   - location – код расположения зуба;
   - quantity – количество оказаний услуги.

9. Price list

   Атрибуты:

   - name – наименовании позиции;
   - price – цена.

2.3 Таблицы сущностей

Ниже представлены таблицы для каждой сущности.

Таблица 2.3.1 – Stuff

Поле	Тип	Ключ	Неопределенное значение	Дополнительно
id	int	Первичный	Нет	Суррогатный, уникальный
name	varchar(50)	Нет	Нет	
surname	varchar(50)	Нет	Нет	
phone	varchar(15)	Альтернативный	Нет	Уникальный
job_id	int	Нет	Нет	
license	varchar(50)	Нет	Да	Уникальный
phone	varchar(15)	Альтернативный	Нет	Уникальный
interest_rate	real	Нет	Нет	

Таблица 2.3.2 – Qualification

Поле	Тип	Ключ	Неопределенное значение	Дополнительно
id	int	Первичный	Нет	Суррогатный, Уникальный
stuff_id	int	Внешний	Нет	
specialization	smallint	Нет	Нет	
organization	varchar(100)	Нет	Нет	
date	date	Нет	Да	

Таблица 2.3.3 – Job

Поле	Тип	Ключ	Неопределенное значение	Дополнительно
id	int	Первичный	Нет	Суррогатный, Уникальный
job_name	varchar(100)	Альтернативный	Нет	Уникальный
daily_salary	numeric	Нет	Нет	

Таблица 2.3.4 – Patient

Поле	Тип	Ключ	Неопределенное значение	Дополнительно
id	int	Первичный, внешний	Нет	Суррогатный, уникальный
name	varchar(50)	Нет	Нет	
surname	varchar(50)	Нет	Нет	
phone	varchar(15)	Альтернативный	Нет	Уникальный

Таблица 2.3.5 – Medical card

Поле	Тип	Ключ	Неопределенное значение	Дополнительно
id	Int	Первичный	Нет	Суррогатный, уникальный
name	Varchar(50)	Нет	Нет	
surname	Varchar(50)	Нет	Нет	
phone	Varchar(15)	Альтернативный	Нет	Уникальный

Таблица 2.3.6 – Treatment

Поле	Тип	Ключ	Неопределенное значение	Дополнительно
id	int	Первичный	Нет	Суррогатный, уникальный
visit_id	int	Внешний	Нет	
code	int	Внешний	Нет	
location	smallint	Нет	Да	
quantity	smallint	Нет	Нет	

Таблица 2.3.7 – Stuff workdays

Поле	Тип	Ключ	Неопределенное значение	Дополнительно
id	int	Первичный	Нет	Суррогатный, уникальный
stuff_id	int	Внешний	Нет	
date	date	Нет	Нет	

Таблица 2.3.8 – Price list

Поле	Тип	Ключ	Неопределенное значение	Дополнительно
id	int	Первичный	Нет	Суррогатный, уникальный
name	varchar(255)	Альтернативный	Нет	Уникальный
price	numeric	Нет	Нет	

Таблица 2.3.9 – Visit

Поле	Тип	Ключ	Неопределенное значение	Дополнительно
id	int	Первичный	Нет	Суррогатный, уникальный
patient_id	int	Внешний	Нет	
doctor_id	int	Внешний	Нет	
date	date	Нет	Нет	
time	time	Нет	Нет	
room	smallint	Нет	Нет	
receipt	text	Нет	Да	

2.4 Обоснование правил связи сущностей

Таблица 2.4.1 – описание связей между сущностями

Связь		Кардинальность		
Родитель	Потомок	тип	Макс.	Мин.
Stuff	Qualification	Идентифицирующая	1:N	M-O
Job	Stuff	Идентифицирующая	1:N	M-O
Stuff	Stuff Workdays	Идентифицирующая	1:N	M-O
Patient	Medical card	Идентифицирующая	1:1	M-M
Stuff	Visit	Идентифицирующая	1:N	M-O
Patient	Visit	Идентифицирующая	1:N	M-O
Visit	Treatment	Идентифицирующая	1:N	M-M
Price list	Treatment	Идентифицирующая	1:N	M-O

В следующих таблицах приведены действия для ограничения минимальной кардинальности.

Идентифицирующая связь Stuff – Qualification 1:N, M-O:

Таблица 2.4.2 - Stuff к Qualification

	Действие на Stuff (родитель)	Действие на Qualification (потомок)
Вставка	Без ограничений	Подбор родительской записи
Изменение ключа или внешнего ключа	Каскадное обновление	Запрещено
Удаление	Запрещено	Разрешено

Таблица 2.4.3 - Job к Stuff

	Действие на Job (родитель)	Действие на Stuff (потомок)
Вставка	Без ограничений	Подбор родительской записи
Изменение ключа или внешнего ключа	Каскадное обновление	Запрещено
Удаление	Запрещено	Запрещено

Таблица 2.4.4 - Stuff к Stuff workdays

	Действие на Stuff (родитель)	Действие на Stuff workdays (потомок)
Вставка	Без ограничений	Подбор родительской записи
Изменение ключа или внешнего ключа	Каскадное обновление	Запрещено
Удаление	Запрещено	Разрешено

Таблица 2.4.5 - Patient к Medical card

	Действие на Patient (родитель)	Действие на Medical card (потомок)
Вставка	Подбор дочерней записи	Подбор родительской записи
Изменение ключа или внешнего ключа	Каскадное обновление	Каскадное обновление
Удаление	Запрещено	Запрещено

Таблица 2.4.6 - Stuff к Visit

	Действие на Stuff (родитель)	Действие на Visit (потомок)
Вставка	Без ограничений	Подбор родительской записи
Изменение ключа или внешнего ключа	Каскадное обновление	Запрещено
Удаление	Запрещено	Запрещено

Таблица 2.4.7 - Patient к Visit

	Действие на Patient (родитель)	Действие на Visit (потомок)
Вставка	Без ограничений	Подбор родительской записи
Изменение ключа или внешнего ключа	Каскадное обновление	Запрещено
Удаление	Запрещено	Запрещено

Таблица 2.4.8 - Visit к Treatment

	Действие на Visit (родитель)	Действие на Treatment (потомок)
Вставка	Создание дочерней записи	Подбор родительской записи
Изменение ключа или внешнего ключа	Каскадное обновление	Запрещено
Удаление	Запрещено	Запрещено

Таблица 2.4.9 – Price list к Treatment

	Действие на Price (родитель)	Действие на Treatment (потомок)
Вставка	Без ограничений	Подбор родительской записи
Изменение ключа или внешнего ключа	Каскадное обновление	Запрещено
Удаление	Запрещено	Запрещено

3. Реализация базы данных и приложения

Для создания базы данных была выбрана система управления реляционными базами данных с открытым исходным кодом – PostgreSQL [1]. Данная СУБД поддерживает запросы SQL (реляционные) и JSON (нереляционные). PostgreSQL возможно расширить с помощью собственных типов данных, индексов и функциональных языков.

Сформируем основные требования к разрабатываемому приложению:

- возможность простого перехода между страницами;
- удобные поля для ввода даты и времени приема;
- удобный поиск по врачам, пациентам и процедурам в виде списка;
- кросс-платформенность;

Для разработки приложения был выбран язык Python 3.7.10 [2] с использованием библиотек psycopg2 [3], tkinter [4] и tkcalendar [5]. Python имеет простой синтаксис и большую общественную поддержку, что позволяет ускорить разработки приложения. Psycopg2 – интерфейс взаимодействия языка Python и PostgreSQL, позволяющий безопасно работать с многопоточными приложениям. Tkinter – кросс-платформенная событийно-ориентированная графическая библиотека на основе средств Tk, предназначенная для организации диалогов с помощью оконного графического интерфейса. TkCalendar – модуль, предоставляющий виджеты ввода даты для Tkinter.

3.1 Реализация интерфейса

В листинге 1 представлен класс App приложения с методом switch_frame, принимающий любой объект Frame из модуля tkinter и создающий новый Frame, удаляя предыдущий. На рисунке 3 изображена главная страница приложения.

Листинг 1 – класс приложения

```
class App(tk.Tk):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self._frame = None
```

Продолжение листинга 1.

```
        self.title_font = tkfont.Font(family='Helvetica',
      size=18, weight="bold", slant="italic")
        self.eval("tk::PlaceWindow . center")
        self.geometry(f"{W}x{H}-{W_BIAS}+0")
        self.configure(background=BG)
        self.focus()
        self.switch_frame(MainPage)

    def switch_frame(self, frame_class):
        new_frame = frame_class(self)
        if self._frame is not None:
            self._frame.destroy()
        self._frame = new_frame
        self._frame.pack_propagate(0)
        self._frame.pack()
```

Листинг 1 – класс приложения

В листинге 2 представлен класс реализации Frame из модуля tkinter на примере главного меню (рисунок 3) приложения с двумя кнопками: перейти к калькулятору зарплат и к меню внесения нового визита.

Листинг 2 – класс главной страницы

```
class MainPage(tk.Frame):
    def __init__(self, master):
        tk.Frame.__init__(self, master)
        self.master = master
        label = tk.Label(self, text='Главная страница', bg=LBL_BG)
        label.grid(row=0, column=2, pady=30, padx=30)

        calc_button = tk.Button(
            self, text='калькулятор зарплат',
            bg=BG, background=BG,
            command=lambda: self.master.switch_frame(CalcPage),
            highlightbackground=BTN_BG, height=3,
        )
        calc_button.grid(row=1, column=1, padx=30, pady=30)
        insert_visit_button = tk.Button(
            self, text='внести визит',
            bg=BG, background=BG,
            command=lambda: self.master.switch_frame(InsertVisitPage),
            highlightbackground=BTN_BG, height=3,
        )
        insert_visit_button.grid(row=1, column=3, padx=30, pady=30)
```

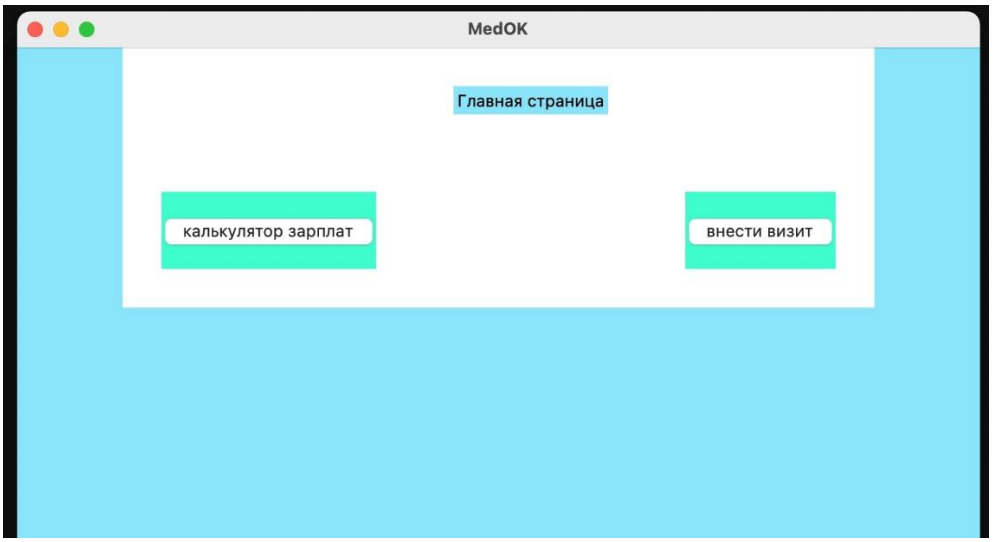

Рисунок 3 – главное меню приложения

Страница расчета зарплат (рисунок 4) реализована аналогично главной с добавлением форм ввода данных.

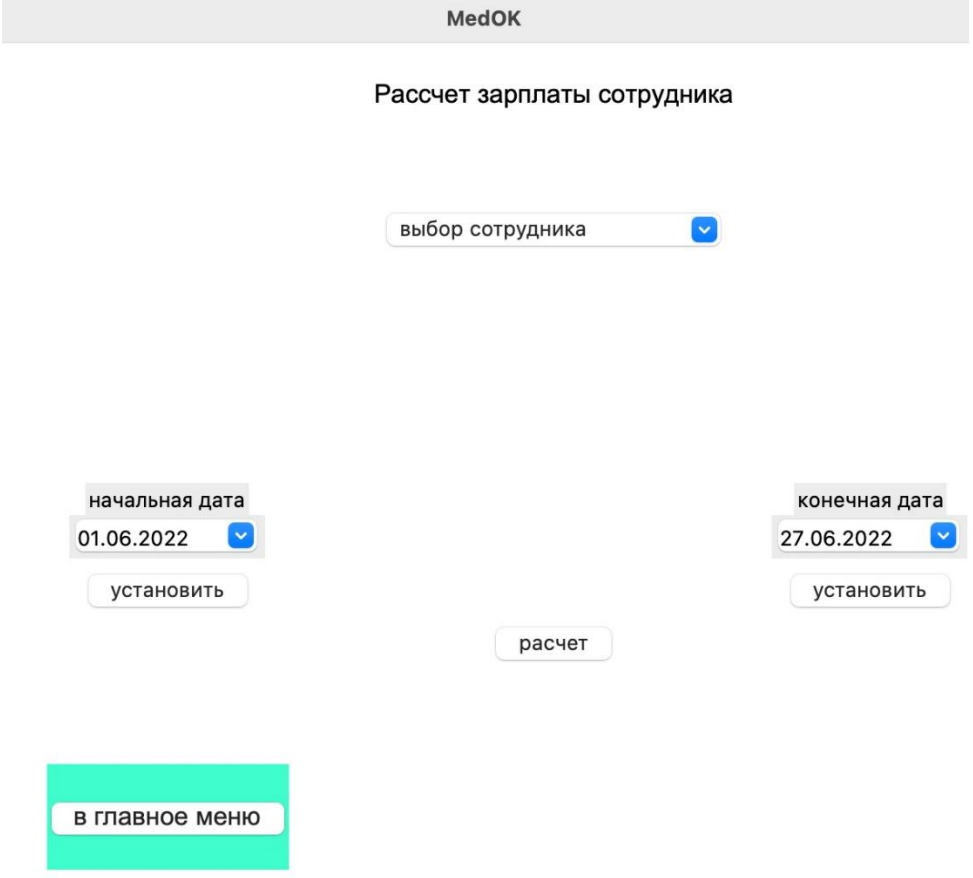

Рисунок 4 – страница расчета заработной платы

Выбор сотрудника представлен в виде выпадающего списка с именами и должностями (рисунок 5) и реализован с помощью OptionMenu (листинг 3).

Листинг 3 – реализация выпадающего меню с сотрудниками

```
def set_drop_menu(self):
    stuff = get_table(cls=Stuff)
    job = get_table(cls=Job)
    names = [s.get_name(job) for s in stuff.values()]
    name_to_id = dict(zip(names, stuff.keys()))
    clicked = StringVar(self)
    clicked.set('выбор сотрудника')

    def on_select(choice):
        text = clicked.get()
        name = text
        self.stuff_id = int(name_to_id.get(name))
        st = stuff.get(self.stuff_id)
        self.salary = job[st.job_id].daily_salary
        text = f'сотрудник: {name}\nсмена: {self.salary}'
        if st.interest_rate != 0:
            self.interest_rate = st.interest_rate
            text += f', {round(self.interest_rate*100, 2)}%'

        label.config(text=text)
    drop = OptionMenu(self, clicked, *names, command=on_select)
    drop.grid(row=1, column=1, padx=10, pady=40)
    label = Label(self, text=' ')
    label.grid(row=2, column=1, padx=10, pady=40)
```

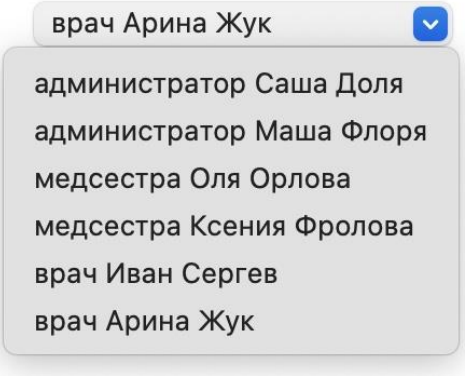

Рисунок 5 – выпадающее окно сотрудников

Внесение начальный и конечной даты для расчета заработной платы реализуется с помощью DataEnrty из модуля TkCalendar, позволяющего вводить дату как в текстовом формате, так и с помощью выпадающего календаря

(листинг 4) и изображен на рисунке 6.

Листинг 4 – ввод даты

```
def set_date_input(self, col, prefix, default_date, is_start=0):
        label = ttk.Label(self, text=prefix)
        label.grid(row=3, column=col, padx=10)
        cal = DateEntry(self, width=12, background='darkblue',
                        year=default_date.year,
                        month=default_date.month,
                        day=default_date.day,
                        foreground='white', borderwidth=2)
        cal.grid(row=4, column=col)
        button = tk.Button(self, text='установить', command=lambda:
    print_date())
        button.grid(row=5, column=col, pady=5, padx=10)
```

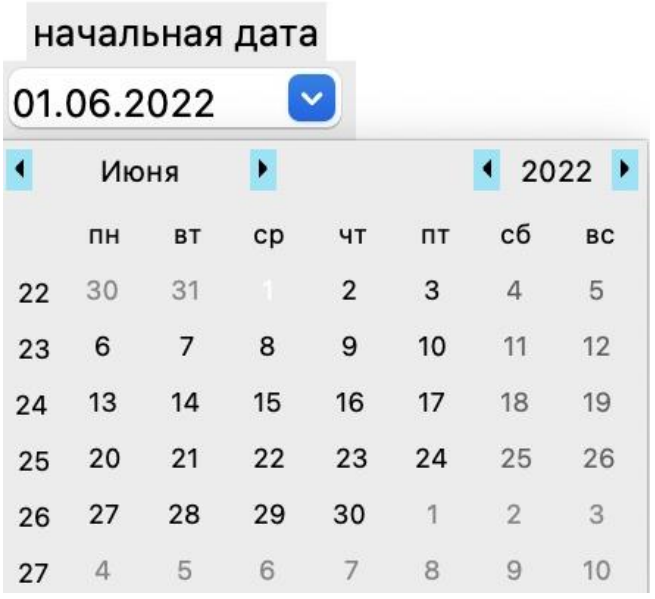

Рисунок 6 – ввод даты

Выпадающие списки сотрудников, пациентов и процедур реализованы аналогично листингу 3.

Сообщения об ошибках внесения реализованы с помощью всплывающих окон messagebox (рисунок 7), базовая проверка представлена в листинге 5.

Листинг 5 – базовая проверка внесения визита

```
TIME_REGEX = '^([0-1]?[0-9]|2[0-3]):[0-5][0-9](:[0-5][0-9])?$'

def _insert_visit():
    if len(self.treatment_list) == 0:
```

Продолжение листинга 5.

```
            messagebox.showerror('Ошибка',  'Нет  ни  одной  процедуры  для
визита')
            return
        if not self.patient_id:
            messagebox.showerror('Ошибка', 'Внесите пациента')
            return
        if not self.doctor_id:
            messagebox.showerror('Ошибка', 'Внесите врача')
            return
        if not re.fullmatch(TIME_REGEX, self.time.get()):
            messagebox.showerror('Ошибка', 'Введите корректно
                                 время в формате hh:mm:ss')
            return
```

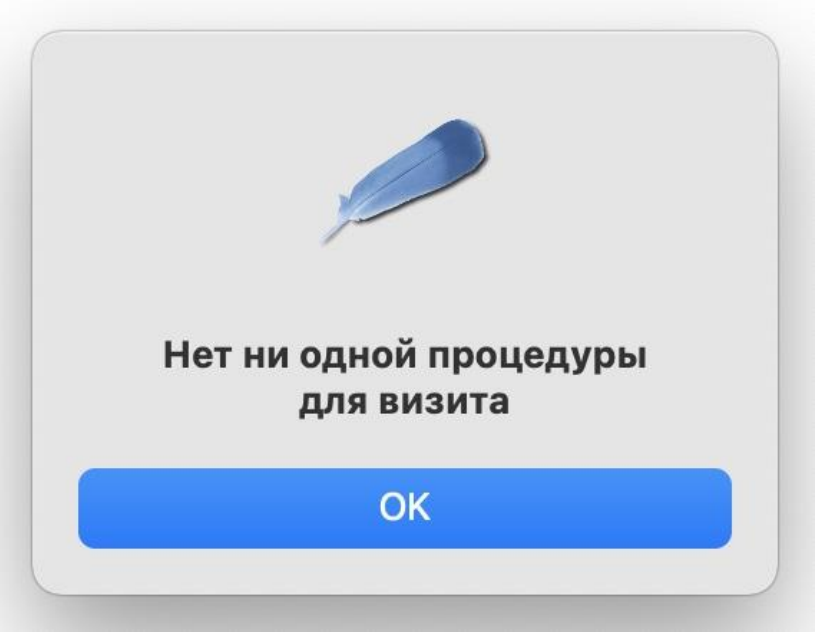

Рисунок 7 – пример всплывающего окна с ошибкой

3.2 Реализация запросов к базе данных

В листинге 6 изображено подключение к базе данных с помощью языка Python и библиотеки psycopg2.

Листинг 6 – подключение к базе данных

```
config = {
    'host': 'localhost',
    'user': 'postgres',
    'password': '123',
    'database': 'clinic'
}
```

```
try:
```
Продолжение листинга 6.
```
    conn = psycopg2.connect(**config)
except Exception as ex:
    print(f'Cannot connect: {ex}')
```

Классы модели данных наследуются от обобщенного класса Entity, имеющего методы get_data для получения словаря из полей для генерации строки sql.

В листинге 7 представлена функция get_table, позволяющая получить список всех сущностей таблицы в ООП представлении из базы данных, передав соединение и cls – класс, полностью соответствующий табличному представлению.

Листинг 7 – функция get_table

```
def get_table(cls, conn=get_connection()):
    table_name = cls.__name__.lower()
    query = f'select * from {table_name}'
    try:
        with conn.cursor() as cur:
            cur.execute(query)
            conn.commit()
            rows = cur.fetchall()
            entities = [cls(*r) for r in rows]
            table = dict()
            for e in entities:
                table[e.id] = e
            return table
    except Exception as ex:
        conn.rollback()
        print(f"Exeption select: {ex} for table {table_name}")
        return None
```

В листинге 8 представлена функция insert, позволяющая вставить любой экземпляр, реализующего базовый класс Entity за счет генерации строки SQL исходя из полей класса.

Листинг 8 – функция insert

```
def insert(entity: Entity, conn=get_connection()):
    table_name = entity.__class__.__name__.lower()
    query = f'insert into {table_name}'
    d = entity.get_data()
    fields = d.keys()
```

```
    values = list(d.values())
```

Продолжение листинга 8.

```
    query += ' (' + ','.join(fields) + ')'
    query += ' values (' + ','.join(['%s'] * len(values)) + ');'
    try:
        with conn.cursor() as cur:
            cur.execute(query, values)
            conn.commit()
        return 0
    except Exception as ex:
        conn.rollback()
        print(f"Exeption in insert: {ex} for table {table_name}+
            f”with entity {entity}")
        return ex
```

Пример реализации триггера на вставку посещения изображен в листинге 9: только врач может вести прием и, более того, только в рабочие дни.

Листинг 9 – триггер на вставку посещения

```
create or replace function check_stuff()
    returns trigger
as $check_stuff$
begin
    if not exists(select id, job_id from stuff where id=new.doctor_id and
job_id=3) then
        raise exception 'Only doctor can hold a visit';
    end if;
    if new.date not in(
        select date from stuff_workdays s where s.stuff_id=new.doctor_id
    ) then
        raise exception 'Doctor cannot hold a visit on a non-working day';
    end if;
    return new;
end;
$check_stuff$ language plpgsql;
```

Реализация M-M на примере связи Patient-Medical Card показана в листинге 10. Ключевое слово DEFERRABLE определяет, можно ли отложить данное условие. В данном примере, откладывается установление внешнего ключа.

```
create table medical_card(
    id              int             primary key,
    sex             char(1)         not null check (sex in ('M', 'F')),
    blood_type      nchar(3)        not null check (blood_type in (
                                                    'O+', 'O-
',
```

Продолжение листинга 10.

```
                                                                'A+', 'A-
',
                                                                'B+', 'B-
',
                                                                'AB+',
'AB-')),
    birth_date      date            not null,
    allergy         text            null,
    diseases        text            null,
    medicines       text            null
);


create table patient(
    id              int             primary key,
    name            varchar(50)     not null,
    surname         varchar(50)     null,
    phone           varchar(15)     not null,
    unique(phone)
);


alter table medical_card
        add foreign key(id) references patient (id)
            DEFERRABLE INITIALLY DEFERRED;

alter table patient
        add foreign key(id) references medical_card (id)
            DEFERRABLE INITIALLY DEFERRED;
commit;
```

4. Тестирование

В ходе тестирование необходимо проверить осуществление расчета заработной платы сотрудникам в соответствии с их должностью и расписанием, внесение визита в соответствии с рабочим расписанием.

Например, введем в базу данных врача Арину Жук с процентной ставкой 25%, и ее рабочие с помощью генерации серии дат начиная 2 июня и заканчивая 23 с интервалом в два дня (листинг 10). Ожидаемый результат, одиннадцать смен по три тысячи и суммарно за визиты две тысячи.

Листинг 10 – ввод произвольных данных

```
insert into job (id, daily_salary, job_name) values
    (3, 3000, 'врач');
commit;

insert into stuff (id, name, surname, job_id, license, phone,
interest_rate) values
        (6, 'Арина', 'Жук', 3, 'DOC123-4124', '89617391777', 0.25);
commit;

insert into stuff_workdays(stuff_id, date)
select 6, * from generate_series('2022-06-02'::date, '2022-06-23'::date,
'2 day'::interval);

insert into price_list values
    (1, 'анестезия', 1000),
    (2, 'удаление зуба', 2500),
    (3, 'лечение кариеса', 3000),
    (4, 'установка коронки', 5000);
commit;
insert into visit(patient_id, doctor_id, date) values
        (2, 6,'2022-06-02');
insert into treatment (visit_id, code, quantity) values
    (2, 1, 2),
    (2, 3, 2);
commit;
```

На рисунке 8 изображено главное меню приложения, с помощью которого можно перейти на страницы калькулятора и внесения визита.

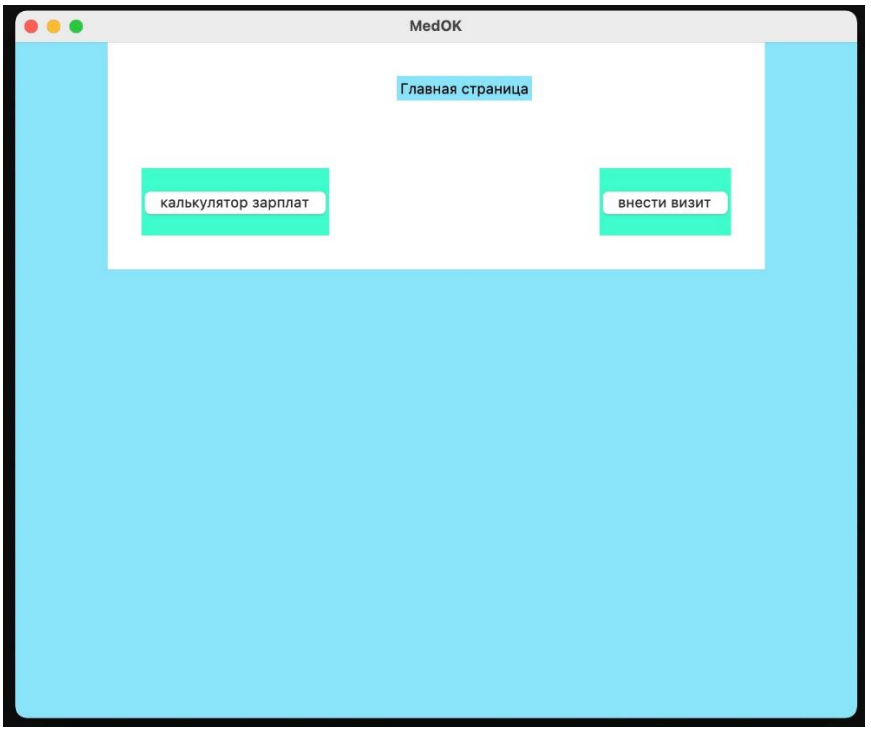

Рисунок 8 – главная страница

Перейдем в калькулятор зарплат, изображенный на рисунке 9. Получаем правильный результат

Рисунок 9 – страница калькулятора зарплат

Теперь внесем визит, на котором было вылечено 10 кариесов, с помощью страницы внесения через графический интерфейс (рисунок 10).

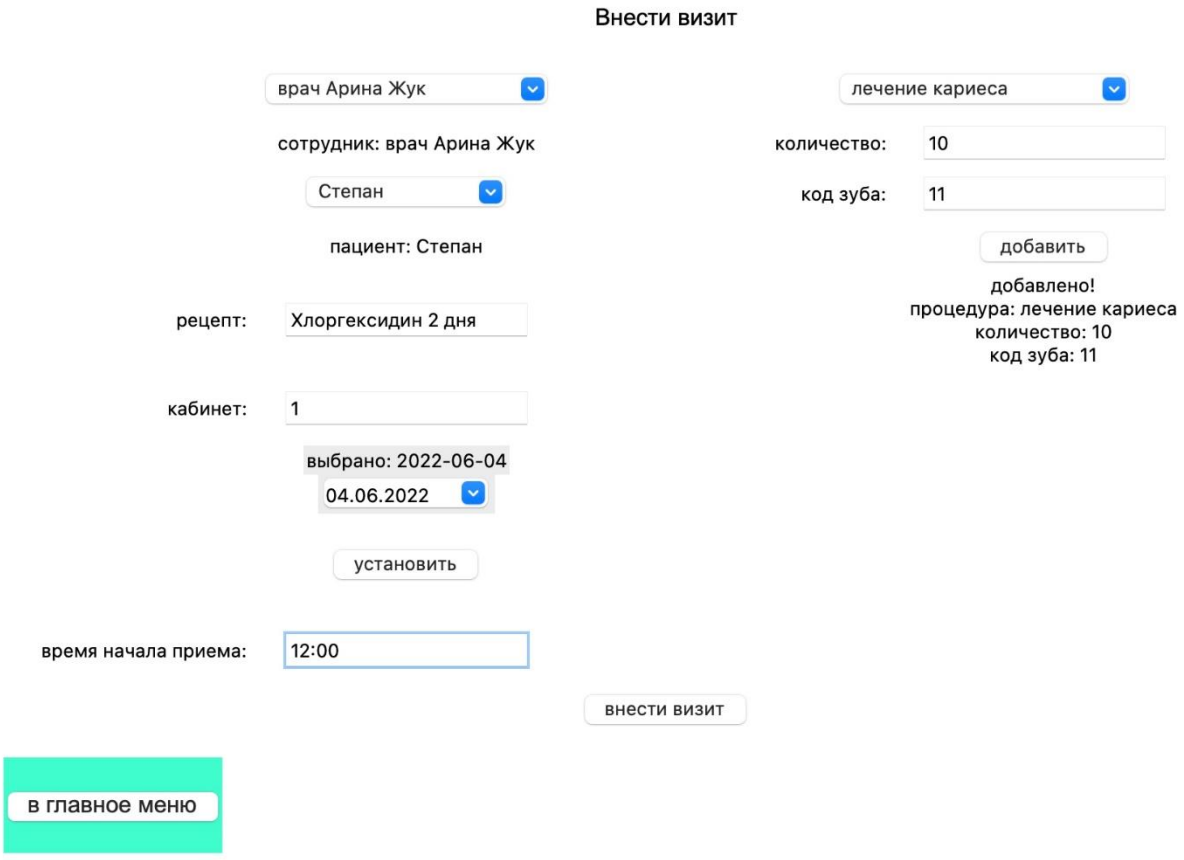

Рисунок 10 – страница внесения визита

Вернемся в калькулятор зарплат, чтобы убедится в изменение заработной платы с учетом нового визита (рисунок 11).

Рисунок 11 – изменения в заработной плате

ЗАКЛЮЧЕНИЕ

В итоге выполнения данной курсовой работы были разработаны реляционная модель и модель “сущность-связь” стоматологической клиники, создана база данных на СУБД PostgreSQL. Было разработано кросс-платформенное оконное приложение на основе событийно-ориентированной графической библиотеке с помощью средств Tk.

Приложение учитывает специфику вводимых и выводимых данных, производит базовую проверку вводимых значений и, в случае ошибки, выводит ее на экран.

В дальнейшем, предполагается ввести аутентификацию и разделению функциональности для разных пользователей. Например, администратор лишь сможет вносить визиты, а бухгалтер – проводить расчет зарплат.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация PostgreSQL. – URL: https://www.postgresql.org/docs/ (дата обращения 20.06.2022)
2. Документация Python. – URL: https://www.python.org/doc/ (дата обращения 20.06.2022)
3. Документация Psycopg2. – URL https://www.psycopg.org/docs/ (дата обращения 20.06.2022)
4. Документация Tkinter. – URL: https://docs.python.org/3/library/tkinter.html (дата обращения 20.06.2022)
5. Документация TkCalendar. – URL: https://github.com/j4321/tkcalendar#documentation (дата обращения 20.06.2022)