

**МІНІСТЕРСТВО ОСВІТИ И НАУКИ УКРАЇНИ**  
**ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І. МЕЧНИКОВА**  
**Факультет математики, фізики та інформаційних технологій**  
**Кафедра математичного забезпечення комп'ютерних систем**

**КУРСОВИЙ ПРОЕКТ**  
з дисципліни  
**«Організація баз даних»**  
на тему:  
**«Інформаційна система облік тварин»**

студента Єлєсіна О. О. III курсу

групи \_\_\_\_\_

спеціальності \_\_\_\_\_

\_\_\_\_\_  
(Прізвище, ім'я та по батькові)

Керівник: \_\_\_\_ ст. викл. Розновець О.І. \_\_\_\_\_

Захищено « \_\_\_\_ » \_\_\_\_\_ 202\_\_ р.

з оцінкою \_\_\_\_\_

Комісія:

\_\_\_\_\_  
(ПІБ)

\_\_\_\_\_  
(Підпис)

\_\_\_\_\_  
(ПІБ)

\_\_\_\_\_  
(Підпис)

\_\_\_\_\_  
(ПІБ)

\_\_\_\_\_  
(Підпис)

**Одеса - 2022**

## **АНОТАЦІЯ**

Мета даного курсового проекту – проектування і реалізація інформаційної системи обліку тварин. Користувачами цієї системи являються робітники умовної ферми. Реалізація виконана з використанням мови Python3 і СУБД PostgreSQL. Архітектура системи відповідає триланковій моделі клієнт-сервер.

У розробленій системі передбачена можливість створення, обробки та редагування заявок кількох видів, переглядання графіків та збереження різноманітної корисної інформації про тварин.

В інформаційній системі реалізований захист від несанкціонованого доступу та здійснено розмежування повноважень різних категорій користувачів. Результатом курсового проектування є інформаційна система обліку тварин зі зручним інтерфейсом.

## ЗМІСТ

АНОТАЦІЯ .....	2
ЗМІСТ .....	3
ВСТУП.....	4
1 ПОСТАНОВКА ЗАДАЧІ.....	5
2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ.....	9
3 ІНФОРМАЦІЙНА МОДЕЛЬ ПРЕДМЕТНОЇ ОБЛАСТІ .....	11
4 ПРОГРАМНА МОДЕЛЬ ЗАСТОСУНКА .....	14
5 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	15
6 СТВОРЕННЯ БАЗИ ДАНИХ.....	16
7 ЗАПИТИ ДЛЯ РОЗВ'ЯЗАННЯ ПОСТАВЛЕНИХ ЗАДАЧ.....	17
8 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ .....	26
9 БЕЗПЕКА ІНФОРМАЦІЙНОЇ СИСТЕМИ .....	31
10 ІНСТРУКЦІЯ КОРИСТУВАЧА .....	37
ВИСНОВКИ.....	43
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....	44
ДОДАТОК А. Опис сутностей та їх властивостей .....	45
ДОДАТОК Б. Схема бази даних .....	50
ДОДАТОК В. Ієрархії елементів системи .....	51
ДОДАТОК Г. Запити на створення таблиць БД .....	53
ДОДАТОК Д. Запити на створення тригерів .....	57
ДОДАТОК Е. Запити на створення функцій.....	58
ДОДАТОК Ж. Запити на створення представлень.....	74

## ВСТУП

У сучасному світі цифрові технології все більше проникають у наше повсякденне життя і сприяють появі нових інформаційних систем. Грамотно спроектована та стабільно працююча інформаційна система здатна спростити та прискорити виконання багатьох повсякденних задач.

Ведення обліку тварин на будь-якому підприємстві або в будь-якій установі являє собою трудомісткий, тривалий і ресурсозатратний процес, при якому необхідно постійно стежити за зміною стану тварин. Цей процес обтяжується людським фактором, оскільки неможливо відстежити вручну всі зміни великої кількості тварин. Для автоматизації обліку тварин застосовуються спеціалізовані інформаційні системи.

Мета даного курсового проекту є проектування і реалізація інформаційної системи обліку тварин, яка дозволить отримувати інформацію про актуальний стан тварин, створювати та оброблювати заявки наступних типів: на списання, на зміну корму та на огляд.

Для досягнення такої мети потрібно розв'язати наступні задачі:

- проаналізувати предметну область, виокремити користувачів системи;
- виконати проектування бази даних;
- обґрунтувати вибір засобів та технологій розробки;
- розробити базу даних на основі системи керування базами даних PostgreSQL;
- розробити клієнтську частину застосунку;
- розробити серверну частину застосунку;
- протестувати програмний застосунок.

## 1 ПОСТАНОВКА ЗАДАЧІ

### Задачі інформаційної системи

До основних задач інформаційної системи відносяться наступні:

- додавання та списання тварин;
- перегляд тварин певного стану(хворі, голодні і т.д.);
- перегляд різноманітних графіків(темпи росту, залежність тривалості життя від типу корму і т.д.);
- подання, обробка та видалення заявок;
- забезпечити захищеність системи за допомогою системи аутентифікації;
- розмежити права та доступ різних типів користувачів.

### Типи користувачів системи

В інформаційній системі передбачено три основних типи користувачів та один технічний. Кожен з типів має свої можливості та вимоги до системи. Типи користувачів перераховано нижче:

- адміністратор - виконує зміну даних, керує списками користувачів підприємства та їх заявками;
- ветеринар – проводить огляд, лікування та подання заявок на списання тварин. Змінює тип кормів. Має доступ до статистики смертності за віком та типом корму;
- робочий – годує, додає та здійснює зважування тварин. Має статистику темпу зростання залежно від віку чи типу корму. Подає заявки на огляд та зміну типу корму;
- завідувач підприємства – приймає та видаляє співробітників. Також оброблює заявки по списанню тварин. Може переглядати інформацію про співробітників та тварин;
- логін – технічна роль, яка використовується тільки для автентифікації користувачів при першому підключенні до системи.

## Список задач користувачів системи

Задачі користувачів ІС обліку тварин, із вказанням вхідних та вихідних даних наведені в табл. 1.1.

Таблиця 1.1 – Список задач користувачів ІС «Облік тварин»

Номер	Задача	Вхідні дані	Вихідні дані
<b>Ветеринар</b>			
B1	Подати заявку на списання	Співробітник, дата, номер тварини, тип заявки – «списання»	Нова заявка на списання тварин
B2	Переглянути список хворих тварин	статус «хворий»	Вибірка хворих тварин
B3	Переглянути список здорових тварин	статус «здоровий»	Вибірка здорових тварин
B4	Змінити тип корму тварини	Співробітник, дата, номер тварини, тип корму	Оновлені дані про тип корму для тварини
B5	Переглянути залежність смертності від віку	тип тварини, статус «списаний»	Графік залежності смертності від віку тварин
B6	Переглянути тривалість життя від типу корму	тип тварини, статус «здоровий», тип корму	Графік залежності тривалості життя від типу корму
B7	Провести огляд тварини	ID співробітника, дата, номер тварини, дані про стан тварини	Оновлені дані про стан тварини
<b>Робочий</b>			
P1	Годувати тварину	Співробітник, дата, час, номер тварини, тип корму, статус «годований»	Оновлені дані про ситість тварини

## Продовження таблиці 1.1

Номер	Задача	Вхідні дані	Вихідні дані
P2	Здійснити контрольне зважування тварини	Співробітник, дата, номер тварини, вага	Оновлені дані про вагу тварини
P3	Подати заявку на огляд тварини	Співробітник, дата, номер тварини, тип заявки – огляд	Нова заявка на огляд тварини
P4	Додати тварину	номер, дата, вік, вага, тип корму, стан, тип тварини	Нова тварина
P5	Переглянути список ситих тварин	тип тварини, статус «ситий»	номер, вік, вага, тип корму, стан, тип тварини
P6	Переглянути список голодних тварин	тип тварини, статус «голодний»	номер, вік, вага, тип корму, стан, тип тварини
P7	Переглянути темп зростання	Номер тварини	Графік темпу зростання
P8	Переглянути залежність ваги від типу корму	Номер тварини	Графік залежності ваги від типу корму
P9	Подати заявку на зміну типу корму	Співробітник, дата, номер тварини, тип заявки - "зміна типу корму"	Нова заявка про зміну типу корму
<b>Адміністратор</b>			
A1	Редагувати співробітника	Співробітник, що змінюється, нові логін, пароль, посада, стан «працює», ПІБ, дата народження	Оновлена інформація про співробітника
A2	Змінити заявку на списання	Співробітник, дата, номер тварини, тип заявки – «списання»	Оновлена заявка про списання
A3	Змінити заявку на огляд	Співробітник, дата, номер тварини, тип заявки – «огляд»	Оновлена заявка про огляд

## Продовження таблиці 1.1

Номер	Задача	Вхідні дані	Вихідні дані
A4	Змінити заявку про зміну типу корму	Співробітник, дата, номер тварини, тип заявки - "зміна типу корму"	Оновлена заявка про зміну типу корму
A5	Видалити заявку	Заявка	Відсутні
A6	Переглянути список співробітників	Відсутні	ПІБ, дата початку роботи, стаж, посада
<b>Завідувач підприємства</b>			
31	Видалити співробітника	Співробітник, стан – «не працює»	Звільнений співробітник
32	Створити співробітника	Логін, пароль, Посада, ПІБ, дата народження	Новий співробітник
33	Списання тварини	дата, номер тварини, Статус – «списаний»	Відсутні
34	Переглянути списаних тварини	Статус – «списаний»	номер, вік, вага, тип корму, тип тварини
35	Перегляд діючих працівників	ID співробітника, стан – «працює»	Список співробітників
36	Перегляд не списаних тварин	Номер тварини, статус – «здоровий» + статус «хворий»	Список тварин



## 2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Для розв'язання поставлених задач була використана двуланкова архітектура клієнт-сервер, яка передбачає застосування наступних компонентів: клієнтський застосунок, який підключений до бази даних. Схема двуланкової архітектури наведена на рисунку 2.1.

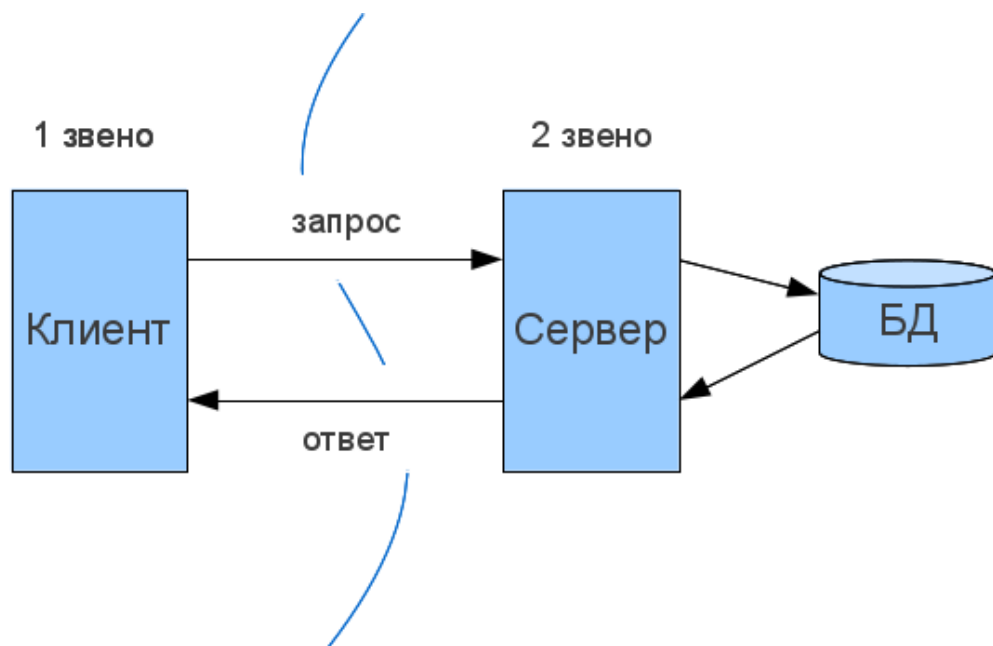


Рисунок 2.1 – Схема двуланкової архітектури

У якості шаблону проектування обрано MVP (Model, View, Presenter), де модель представляє дані та методи їх обробки, контролер забезпечує зв'язок користувача з застосунком, а Вид отримує дані від Представлення та відображає в інтерфейсі користувача. Схему взаємодії між моделлю, представленням та видом наведено на рисунку 2.2.

При використанні шаблону MVP, розробленого для поліпшення відділення логіки від відображення, ІС також ділиться на три окремі блоки:

1) Модель, яка здійснює маніпулювання даними застосунка, надання даних Представленню і реагування на команди Представлення шляхом зміни свого стану;

2) Вид, яке відповідає за відображення даних користувачеві (звертаючись при цьому за цими даними до Представника), а також за отримання від користувача команд для роботи з даними і відправку цих команд Представнику;

3) Представлення, яке відповідає за перетворення команд користувача, одержуваних від Виду, в набір дій над Моделлю з метою її зміни, а також виконує функції зміни Виду при зміні стану Моделі, будучи своєрідним «посередником» між Моделлю і Видом.

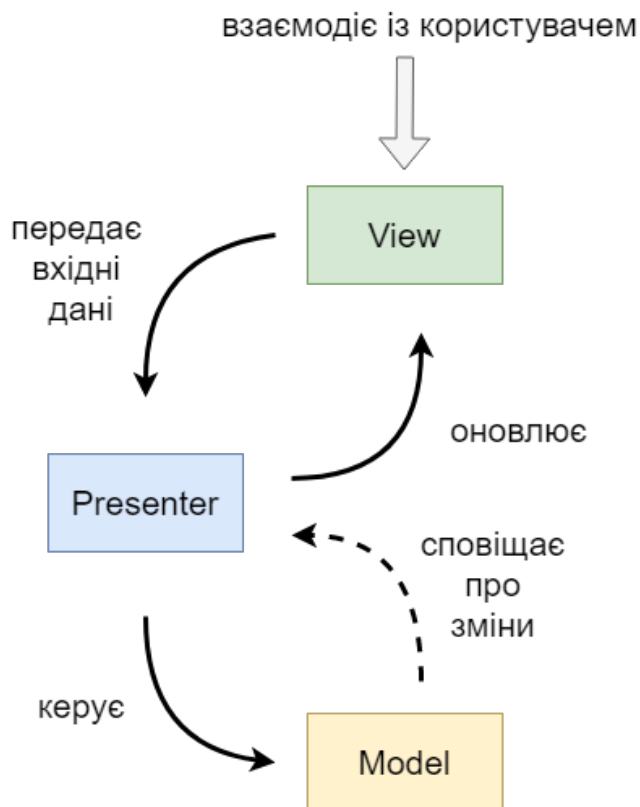


Рисунок 2.2 – Модель MVP

### 3 ІНФОРМАЦІЙНА МОДЕЛЬ ПРЕДМЕТНОЇ ОБЛАСТІ

В інформаційній системі таксі слід виділити наступні сутності:

- співробітник – містить номер співробітника, логін, хешований пароль, повне ім'я та прізвище, дату початку роботи, посаду та значення значення зарплати;
- тварина – номер тварини, тип тварини, тип корму та іншу інформацію, яка характеризує тварину;
- тип тварини – перелік можливих типів тварин;
- заява – містить перелік заявок;
- уточнення до заяви – містить номери тварин, які указані у заявці з певним номером;
- тип заяви – перелік можливих типів заявок;
- ситість тварини – містить інформацію про годування тварин;
- корм – містить інформацію про доступні корма;
- історія оглядів – інформація про огляд кожної тварини;
- динаміка зростання – фіксує вагу кожної тварини на протязі часу.

Детальніше сутності та їх властивості з описом обмежень, що потрібні для розв'язання поставлених задач, наведено в таблиці у додатку А.

Між сутностями у базі даних наявний один тип зв'язку - «один-до-багатьох». Розглянемо кожен зв'язок:

- Між співробітником та заявою. Кожен співробітник може мати багато заявок, але кожна заявка була створена одним співробітником. Для формалізації зв'язку у таблиці employee є зовнішній ключ, який посилається на первинний ключ таблиці application;
- між співробітником та історією оглядів. Кожен співробітник може робити багато оглядів, але кожен огляд був зроблений одним співробітником. Для формалізації зв'язку у таблиці employee є зовнішній ключ, який посилається на первинний ключ таблиці inspection\_history;

- між співробітником та динамікою зростання. Кожен співробітник може зробити декілька записів фіксування ваги, але кожен запис був зроблений одним співробітником. Для формалізації зв'язку у таблиці `employee` є зовнішній ключ, який посилається на первинний ключ таблиці `dynamic_growth`;
- між співробітником та ситістю тварини. Кожен співробітник може погодувати декілька тварин, але кожна тварина була поведена одним співробітником. Для формалізації зв'язку у таблиці `employee` є зовнішній ключ, який посилається на первинний ключ таблиці `satiety_animal`;
- між заявкою та уточненням до заявки. В кожній заявці може бути декілька тварин, але кожна тварина міститься у одній заявці. Для формалізації зв'язку у таблиці `application` є зовнішній ключ, який посилається на первинний ключ таблиці `clarification_to_app`;
- між заявкою та типом заявки. Кожен тип заявки може бути у декількох заявках, але кожна заявка може бути лише одного типу. Для формалізації зв'язку у таблиці `type_application` є зовнішній ключ, який посилається на первинний ключ таблиці `application`;
- між твариною та уточненням до заявки. Кожна тварина може знаходитись в уточненні до заявки. Для формалізації зв'язку у таблиці `animal` є зовнішній ключ, який посилається на первинний ключ таблиці `clarification_to_app`;
- між твариною та історією оглядів. Кожна тварина може бути оглянута. Для формалізації зв'язку у таблиці `animal` є зовнішній ключ, який посилається на первинний ключ таблиці `inspection_history`;
- між твариною та динамікою зросту. Кожна тварина може бути зважена, але кожен запис відповідає лише одній тварині. Для формалізації зв'язку у таблиці `animal` є зовнішній ключ, який посилається на первинний ключ таблиці `dynamic_growth`;
- між твариною та ситістю. Кожна тварина може бути нагодована, але кожному запису відповідає одна тварина. Для формалізації зв'язку у

таблиці `animal` є зовнішній ключ, який посилається на первинний ключ таблиці `satiety_animal`;

- між кормом та твариною. Кожним типом корму може харчуватися декілька тварин, але кожна тварина харчується тільки одним типом корму. Для формалізації зв'язку у таблиці `feed` є зовнішній ключ, який посилається на первинний ключ таблиці `animal`;
- між типом тварини та твариною. Може бути декілька тварин одного типу, але кожна тварина належить до одного типу. Для формалізації зв'язку у таблиці `type_animal` є зовнішній ключ, який посилається на первинний ключ таблиці `animal`;

Схему бази даних, що ілюструє сутності та зв'язки між ними, наведено у додатку Б. Ієрархію елементів системи наведено у додатку В.

#### **4 ПРОГРАМНА МОДЕЛЬ ЗАСТОСУНКА**

Адміністратор, завідувач підприємства, ветеринар та робочий користуються єдиним застосунком.

При авторизації у ролі одного з користувачів, застосунок завантажує необхідну сторінку в залежності від ролі користувача. Ці сторінки відображають різну інформацію та інструменти, що потрібні користувачу для вирішення його задач.

Застосунок дозволяє користувачу зайти у систему під будь-якою роллю, для цього потрібно лише авторизуватись у відповідній формі. Програма складається з 4-х вкладок під кожен роль відповідно.

Дотримання шаблону MVP в даному випадку можливе без створювання додаткових класів. Так, клієнт, через UI надсилає запит, який оброблюється відповідним контролером, в залежності від поточної ролі користувача, а далі, контролер обробляє запит, і передає його до моделі, яка в свою чергу виконує запит до сервера бази даних, і отримавши відповідь, надсилає необхідні дані назад до UI.

## 5 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Інформаційна система розроблена у вигляді прикладної програми, взаємодіяти з таким застосунок користувач може запустивши програму на комп'ютері.

Для роботи з базою даних обрано СУБД PostgreSQL, а для розробки програмного застосунка використовується мова Python3.

Сьогодні існує багато систем керування базами даних. Серед них у проекті обрано PostgreSQL з наступних причин:

- реалізує реляційну модель даних, яка є зрозумілою для кінцевого користувача;
- має гнучкий механізм управління правами користувачів БД (ролі);
- підтримує мову plpgsql як розширення стандарту SQL для створення ефективних збережених процедур;
- підтримує курсорні цикли у збережених процедурах, що спрощує програмування.

Для того, щоб взаємодіяти із сервером бази даних із серверу застосунка, була використана бібліотека Psycopg2, яка дозволяє під'єднуватися до бази даних, виконувати запити та зчитувати отриману відповідь у вигляді масиву, який складається із асоціативних масивів (або словників), що дозволяє вже у серверу застосунка звертатися до відповідних стовпчиків вихідної таблиці за іменем, що є значно зручнішим, аніж індексація за номерами.

Для розробки інтерфейсу мною була обрана PyQt. PyQt – це бібліотека, яка дозволяє використовувати фреймворк Qt GUI (GUI – це графічний інтерфейс користувача) у Python. Сам Qt, як відомо, написано на C++. Використовуючи його в Python, ми можемо створювати програми набагато швидше, не жертвуючи значною частиною продуктивності C++.

## 6 СТВОРЕННЯ БАЗИ ДАНИХ

В цьому розділі описано створення основних об'єктів бази даних. Приведемо приклад створення таблиці Application. Повний код створення всіх таблиць БД наведено в додатку В.

```
CREATE TABLE Employee(  
    ID_employee serial not null PRIMARY KEY,  
    Login varchar(100) not null UNIQUE,  
    Password varchar(100) not null,  
    FullName varchar(100) not null,  
    StartDate date not null CHECK(StartDate <= CURRENT_DATE),  
    Position varchar(40) not null CHECK(Position in ('Ветеринар',  
'Рабочий', 'Администратор', 'Заведующий хозяйством')) ,  
    Status varchar(12) not null CHECK(Status in ('Работает', 'Не  
работает'))  
);
```

### Лістинг 6.1 – Створення таблиці Employee

Для того, щоб створити нову таблицю, використовуємо команду CREATE TABLE, після чого треба вказати ім'я. Далі вказуємо схему таблиці, перелічуючи назви стовпчиків та відповідні типи даних, обмеження та значення за замовчуванням. Поле ID\_employee є первинним ключем (PRIMARY KEY) і має тип serial.

NOT NULL при визначенні полів означає, що вони не можуть містити порожні значення.

Ключове слово UNIQUE значить, що поле, або група полів повинні мати унікальне значення, яке не може повторюватися у межах даної таблиці.

ключове слово CHECK використовується для перевірки певних умов, які вказуються у круглих дужках.

Тип даних VARCHAR(100) – це рядок, довжиною максимум 100 символів.



## 7 ЗАПИТИ ДЛЯ РОЗВ'ЯЗАННЯ ПОСТАВЛЕНИХ ЗАДАЧ

Далі представлені запити до бази даних, які використовуються у застосунку для отримання, вставки, оновлення або видалення даних з таблиць. Необхідно зазначити, що більшість запитів до сервера бази даних виконуються через SQL функції сервера бази даних. Розпочнемо з ролі ветеринара.

Для виконання задачі B1, яка полягає у подачі заявки на списання тварини, існує функція `add_application` (додаток E, лістинг E.1). Ця функція також використовується для вирішення задач P3 та P9. Функція приймає параметри:

- Тип функції
- Масив ідентифікаторів тварин
- Повне ім'я співробітника, який подає заявку

Усередині даної функції проводяться всілякі перевірки, після чого з'являється нова заявка обраного типу.

```
SELECT * FROM add_application(%s, %s, %s)
```

Лістинг 7.1 – Запит на подачу нової заявки

Для вирішення задачі B2 і B3, мета яких вивести список хворих і здорових тварин, зроблене представлення `info_animals_sick` (Додаток Ж, лістинг Ж.5) і `info_animals_healthy` (Додаток Ж, лістинг Ж.6). Вони повертають таблиці (із 5 стовбців) хворих тварин і здорових тварин відповідно.

```
def listSickAndHealthyAnimals(isHealthy: bool) -> []:
    with __connection:
        try:
            __cursor = __connection.cursor()
            if isHealthy:
                tmp = sql.SQL(f'SELECT * FROM info_animals_healthy')
            else:
                tmp = sql.SQL(f'SELECT * FROM info_animals_sick')
            __cursor.execute(tmp, ())
            dat = __cursor.fetchall()
            return dat
        except psycopg2.Error as e:
            print(e)
```

Лістинг 7.2 – Функція `listSickAndHealthyAnimals`

Дана функція повертає список здорових чи хворих тварин в залежності від параметру `isHealthy`, який приймає значення `True` чи `False`.

Задача В4 вирішується за допомогою функції `change_feed` (Додаток Е, лістинг Е.12)

```
SELECT * FROM change_feed(%s, %s, %s, %s)
```

### Лістинг 7.3 – Функція зміни типу корма

Усі можливі перевірки проходять всередині функції. У якості параметрів передаються:

- Повне ім'я ветеринара, який оброблює заявку;
- Ідентифікатор заявки;
- Масив ідентифікаторів тварин;
- Масив ідентифікаторів типів корму, які відповідають ідентифікатору тварини та тому ж місці в масиві.

Задача В5 і В6 вирішується однією функцією всередині `Model`.

```
select age, count(*) from animal where write_off is not null group  
by age order by age
```

### Лістинг 7.4 – вибірка даних для задачі В5

```
select title, count(*) from animal join feed on animal.key_feed =  
feed.key_feed where write_off is not null group by title order by  
title
```

### Лістинг 7.5 – вибірка даних для задачі В6

На основі цих вибірок будуть побудовані відповідні графіки.

Задача В7 вирішується за допомогою функції `inspection` (Додаток Е, лістинг Е.11)

```
SELECT * FROM inspection(%s, %s, %s, %s)
```

### Лістинг 7.6 – функція для задачі В7

Усі можливі перевірки проходять всередині функції. У якості параметрів передаються:

- Повне ім'я працівника, який оброблює заявку;
- Ідентифікатор заявки;
- Масив ідентифікаторів тварин;
- Масив станів відповідних тварин.

Задача P1 вирішується за допомогою функції `feed_animal` (Додаток Е, лістинг Е.4)

```
SELECT * FROM feed_animal(%s, %s)
```

#### Лістинг 7.7 – функція годування тварин

У якості параметрів передається список тварин, яких потрібно нагодувати, та співробітник, який це виконує. Усередині функції здійснюються перевірки наявності діючого робочого та наявності тварин, які були передані. Якщо усі перевірки пройдені, додається запис до таблиці `Satiety_animal` для кожної тварини. Треба зазначити, що дата і час кожного запису додається до таблиці автоматично як поточні.

Також, для уникнення помилок, було створено тригер `satiety_func` (Додаток Д, лістинг Д.2), який відповідає за те, щоб було неможливо погодувати списану тварину.

Задача P2 вирішується за допомогою функції `weight_animal` (Додаток Е, лістинг Е.5)

```
SELECT * FROM weight_animal(%s, %s, %s)
```

#### Лістинг 7.8 – функція контрольного зважування тварин

У якості параметрів передаються номер тварини, її вага та ID співробітника, який проводить зважування. Усередині функції проводяться

перевірки наявності діючого робочого, ваги (не може бути менше 0) та наявності тварини з номером, який був переданий. Якщо все добре, додається запис до таблиці `Dynamic_growth` із переданими параметрами та поточною датою.

Задача P3 та P9 вирішується за допомогою функції `add_application` (Додаток Е, лістинг Е.1), описання якої знаходиться вище (Лістинг 7.1). Різниця лише у переданому параметрі `type_application`, який може приймати значення із таблиці `application_type` (Додаток А, таблиця А.1).

Задача P4 вирішується за допомогою функції `insert_animal` (Додаток Е, лістинг Е.2)

```
SELECT * FROM insert_animal(%s, %s, %s, %s, %s, %s)
```

#### Лістинг 7.8 – функція додавання тварини

У якості параметрів передаються:

- ПІБ робітника, який додає тварину;
- тип тварини;
- корм тварини;
- стать тварини;
- вік тварини;
- вага тварини.

По кожному із параметрів проводиться перевірка а саме:

- Співробітник має бути діючим робочим;
- тип тварини має бути в таблиці `type_animal` (Додаток А, таблиця А.1);
- корм має бути в таблиці `feed` (Додаток А, таблиця А.1);
- стать може бути лише 'мужской' або 'женский';
- вік не може бути менше 0;
- вага не може бути менше 0.

Якщо усі перевірки пройдені успішно, то додаються записи одразу у 4 таблиці. Треба зазначити, що при додаванні тварини вважається, що вона є здоровою та нагодованою.

У таблицю `animal` записуємо поточну дату, корм, тип тварини, вік та стать. При цьому, ми отримуємо номер тварини, який знадобиться для запису у наступні таблиці. Додаємо до таблиці `Dynamic_growth` запис із значеннями: номер тварини, поточну дату, вагу та номер робочого.

До таблиці `Satiety_animal` додається запис із значеннями: номер тварини, номер робочого та поточна дата і час.

До таблиці `Inspection_history` додається запис із значеннями: номер тварини, номер ветеринара (обирається випадково із усіх діючих ветеринарів), поточна дата і статус «Здоровое».

Задачі P5 та P6 вирішуються за допомогою представлень `info_animals_hungry` (Додаток Ж, лістинг Ж.3) і `info_animals_not_hungry` (Додаток Ж, лістинг Ж.4)

```
def listHungryAndFed(isHungry: bool):
    with __connection:
        try:
            __cursor = __connection.cursor()
            if isHungry:
                tmp = sql.SQL(f'SELECT * FROM
info_animals_hungry')
            else:
                tmp = sql.SQL(f'SELECT * FROM
info_animals_not_hungry')
            __cursor.execute(tmp, ())
            dat = __cursor.fetchall()
            return dat
        except psycopg2.Error as e:
```

Лістинг 7.9 – функція `listHungryAndFed`

В залежності від значення параметру `isHungry`, приведена функція повертає список голодних або ситих тварин.

Задача P7 вирішується за допомогою функції `get_weighing_history` (Додаток E, лістинг E.10)

```
SELECT * FROM get_weighing_history(%s)
```

Лістинг 7.10 – функція `get_weighing_history`

Данна функція повертає історію зважувань конкретної тварини, номер якої передається у якості параметру.

Задача P8 передбачає будування графіків, тому робимо наступну вибірку

```
select title, AVG(weight) from dynamic_growth
join animal on animal.numberanimal =
dynamic_growth.numberanimal
join feed on feed.key_feed = animal.key_feed
group by title
order by title
```

Лістинг 7.11 – вибірка даних для задачі P8

В ній ми рахуємо середню вагу тварин для кожного типу корму. Такий графік буде корисний тим, що дає уявлення про доречність використання певного типу корму.

Задача A1 вирішується за допомогою функції `updateInfoEmployee` (Додаток E, лістинг E.15)

```
SELECT * FROM updateInfoEmployee(%s, %s, %s, %s, %s, %s)
```

Лістинг 7.12 – функція оновлення даних співробітника

В якості параметрів передаємо наступні дані:

- ID співробітника (перевіряється, чи існує);

- логін співробітника (перевіряється, чи є унікальним);
- пароль;
- ПІБ робітника;
- посада ( перевіряється, чи існує);
- зарплата (не менше 0);

Відзначимо, що під час збереження даних спрацьовує тригер `hash_password` (Додаток Д, лістинг Д.1), який порівнює новий пароль зі старим. У випадку, якщо паролі збігаються, дані не перезаписуються. Треба сказати, що ми не можемо редагувати пароль співробітника, так як пароль зберігається у вигляді хешу. Ми можемо лише змінити його на новий.

Задачі А2, А3, А4 вирішуються функцією `update_application` (Додаток Е, лістинг Е.17)

```
SELECT * FROM update_application(%s, %s)
```

Лістинг 7.13 – функція оновлення заявки

Як параметри передаються номер заявки та оновлений список тварин. Всередині функції перевіряється, чи існують тварини із списку і чи існує така заявка. Якщо все добре, із таблиці `clarification_to_app` видаляється список тварин із вказаним номером заявки і додається оновлений список.

Задача А5 вирішується за допомогою функції `rm_application` (Додаток Е, лістинг Е.6)

```
SELECT * FROM rm_application(%s)
```

Лістинг 7.14 – функція видалення заявки

У якості параметру передається номер заявки. Всередині функції перевіряється, чи існує заявка с таким номером. Після перевірки, видаляємо дані із таблиць `application` і `clarification_to_app` за номером заявки.

Задача А6 вирішується за допомогою наступних вибірок

```
SELECT id_employee, fullname, position, startdate, salary,
       status FROM employees WHERE status='Работает'
```

Лістинг 7.15 – список працюючих робітників

```
SELECT id_employee, fullname, position, startdate, salary, status
       FROM employees WHERE status='Не работает'
```

Лістинг 7.16 – список не працюючих робітників

Задача 31 вирішується за допомогою функції `rm_employee` (Додаток Е, лістинг Е.16)

```
SELECT * FROM rm_employee(%s)
```

Лістинг 7.17 – функція звільнення робітника

У якості параметра передається ID робітника. Всередині функції перевіряється, чи є такий робітник та яку посаду він займає. Якщо це єдиний робітник на цій посаді, то звільнення неможливе.

Задача 32 вирішується за допомогою функції `add_employee` (Додаток Е, лістинг Е.3)

```
SELECT * FROM add_employee(%s, %s, %s, %s, %s)
```

Лістинг 7.18 – функція додавання робітника

Як параметри передаються:

- логін (перевіряється, чи є унікальним);
- пароль;
- ПІБ робітника;
- посада робітника (перевіряється, чи існує така посада);
- зарплата (не може бути менше 0).



Зазначимо, що при додаванні паролю до таблиці employee, спрацьовує тригер hash\_password (Додаток Д, лістинг Д.1), який перетворює пароль на хеш, який неможливо розкодувати. Це забезпечує додаткову надійність.

Задача 33 вирішується за допомогою функції write\_off\_animals (Додаток Е, лістинг Е.18)

```
SELECT * FROM write_off_animals(%s,%s,%s)
```

Лістинг 7.19 – функція списання тварин

У якості параметрів передаються:

- список тварин в заявці на списання (перевіряється, чи існують і знаходяться в заявці);
- номер заявки на списання (перевіряється, чи існує);
- співробітник (перевіряється, чи існує);

У разі успішних перевірок, оновлюється стан кожної тварини, а тобто :

- у таблиці animal оновлюється поле write\_off на поточну дату;
- у таблицю Inspection\_history додається запис із значенням поточної дати і статусом «Списанное».

Задача 34 і 36 вирішуються за допомогою представлень info\_animals\_not\_writeOff (Додаток Ж, лістинг Ж.1) і info\_animals\_writeOff (Додаток Ж, лістинг Ж.2)

```
SELECT * FROM info_animals_not_writeOff
```

Лістинг 7.20 – Представлення не списаних тварин

```
SELECT * FROM info_animals_not_writeOff
```

Лістинг 7.21 – Представлення списаних тварин

Задача 35 вирішується за допомогою вибірок, які були приведені вище (лістинг 7.15, лістинг 7.16).

## 8 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ

Для того, щоб користувачі інформаційної системи мали змогу нею користуватися, необхідно розробити користувацький інтерфейс, та відповідний код для серверу застосунку, який буде отримувати запити користувача, обробляти їх, виконувати запити до серверу бази даних, та повертати запитувану інформацію користувачу.

Т.к. мною використаний PyQt5, далі наведений приклад реалізації форми аутентифікації користувачів:

```
class Ui_Login_form(object):
    def setupUi(self, Login_form):
        Login_form.setObjectName("Login_form")
        Login_form.resize(500, 400)
        Login_form.setMinimumSize(QtCore.QSize(500, 400))
        Login_form.setMaximumSize(QtCore.QSize(800, 700))
        font = QtGui.QFont()
        font.setFamily("sans-serif")
        font.setPointSize(16)
        font.setBold(False)
        font.setWeight(50)
        Login_form.setFont(font)
        Login_form.setStyleSheet("QMainWindow{\n"
"    background-color: #242424;\n"
"    font-family: sans-serif;\n"
"}")
        Login_form.setTabShape(QtWidgets.QTabWidget.Rounded)
        Login_form.setUnifiedTitleAndToolBarOnMac(False)
        self.centralwidget = QtWidgets.QWidget(Login_form)
        self.centralwidget.setStyleSheet("QWidget{\n"
"    background-color: #242424;\n"
"    font-family: Roboto;\n"
"    color: white;\n"
"    font-size: 20px;\n"
"    font-weight: 100px;\n"
"}\n"
"")
        self.centralwidget.setObjectName("centralwidget")
        self.verticalLayout =
QtWidgets.QVBoxLayout(self.centralwidget)
        self.verticalLayout.setObjectName("verticalLayout")
        spacerItem = QtWidgets.QSpacerItem(20, 20,
QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
        self.verticalLayout.addItem(spacerItem)
        self.horizontalLayout_4 = QtWidgets.QHBoxLayout()
```

```

self.horizontalLayout_4.setObjectName("horizontalLayout_4")
    spacerItem1 = QtWidgets.QSpacerItem(40, 20,
QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
    self.horizontalLayout_4.addItem(spacerItem1)
    self.lbl_error = QtWidgets.QLabel(self.centralwidget)
    sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
QtWidgets.QSizePolicy.Fixed)
    sizePolicy.setHorizontalStretch(0)
    sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.lbl_error.sizePolicy().hasH
eightForWidth())
    self.lbl_error.setSizePolicy(sizePolicy)
    self.lbl_error.setStyleSheet("QLabel{\n"
"    color: #973A38;\n"
"    font-size: 18px;\n"
"    background-color: #242424;\n"
"}")

```

### Лістинг 8.1 – Шаблон форми аутентифікації

Тепер, коли у нас є інтерфейс, ми можемо передати введені дані у Presenter, щоб той їх підготував та відправив на обробку у Model

```

class LoginForm(QMainWindow, Ui_Login_form):
    def __init__(self):
        super().__init__()
        self.login = Ui_Login_form()
        self.login.setupUi(self)
        self.setupLoginUI()

self.login.btn_confirm.clicked.connect(self.confirmPushedLogin)

    def setupLoginUI(self):
        self.setWindowTitle('Login form')
        self.setWindowIcon(QIcon('view/icons/page/login-rounded-
right.png'))
        self.login.lineEdit_login.setPlaceholderText('Enter login')
        self.login.lineEdit_password.setPlaceholderText('Enter
password')

```

```

self.login.btn_confirm.setFocus()

def confirmPushedLogin(self):
    login = self.login.lineEdit_login.text()
    passwd = self.login.lineEdit_password.text()
    result = model.loginCheck(login, passwd)
    if result == -1:
        self.login.lbl_error.setText('Ошибка. Такого сотрудника
не существует !')
    elif result == -2:
        self.login.lbl_error.setText('Ошибка. Данный сотрудник
не работает !')
    elif result == -3:
        self.login.lbl_error.setText('Ошибка. Неверный пароль
!')
    else:
        role, position, info = result[0], result[1], result[2]
        login_window.close()
        model.createConnection(role)
        self.main = MainForm()
        self.main.openWindow(role, position, info)
        self.main.show()

def closeEvent(self, event) -> None:
    model.closeConnection()

```

**Лістинг 8.2 – клас в Presenter для форми login**

Тепер подивимось на обробку інформації, яка поступає с форми login.

```
def loginCheck(login:str, passwd:str, user = 'login') -> [str,
str, ()]:
    with __connection:
        try:
            __cursor = __connection.cursor()
            stmt = sql.SQL(f'SELECT password, fullname, position,
status, id_employee FROM employee WHERE login=%s')
            __cursor = __connection.cursor()
        except psycopg2.Error as e:
            print(e)

            __cursor.execute(stmt, (login,))
            dat = __cursor.fetchall()

            if len(dat) == 0:
                # в случае, если логин
                # неправильный, то запрос вернёт пустой массив
                print('No employee')
                return -1
            else:
                if dat[0][3] == 'Не работает': # проверка работает ли
                # сотрудник
                    return -2
                elif hashPassword(passwd) != dat[0][0]:
                    #
                    # проверка хеша пароля
                    return -3

    global role
    if dat[0][2] == 'Ветеринар':
        role = 'vet'
    elif dat[0][2] == 'Рабочий':
        role = 'worker'
    elif dat[0][2] == 'Администратор':
        role = 'administrator'
    elif dat[0][2] == 'Заведующий хозяйством':
        role = 'head_household'
```

```
id = dat[0][4]
fullname = dat[0][1]
position = dat[0][2]
__cursor.close()
return [role, position, getInfoEmployee(id)]

def hashPassword(passwd: str):
    hashed_string = hashlib.sha256(passwd.encode('utf-
8')).hexdigest()
    return hashed_string
```

Можемо бачити, що дана функція оброблює інформацію з форми та порівнює її з даними с таблиці employee. Цікава річ в тому, що перед тим, як зрівнювати паролі, дані с форми необхідно пропустити через функцію hashPassword, яка повертає хеш пароллю.

## 9 БЕЗПЕКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

Безпека на рівні БД забезпечується шляхом створення ролей користувачів і надання їм відповідних привілеїв. В інформаційній системі «Облік тварин» реалізовано п'ять ролей: Адміністратор, Ветеринар, Завідуючий господарством, Робочий та Логін. У таблиці 9.1 наведені привілеї ролей на таблиці та елементи бази даних. При цьому використані наступні позначення:

- I (Insert) – додавання нової інформації до таблиці;
- S (Select) – перегляд вмісту таблиці;
- U (Update) – редагування існуючої інформації;
- D (Delete) – видалення інформації з таблиці;
- EXEC (Execute) – виконання функції.

Таблиця 9.1. – Права доступу користувачів до бази даних

Таблиці					
	Робочий	Ветеринар	Адміністратор	Завідуючий господарством	login
animal	S	SU			
application		SD			
application_type					
clarification_to_app		SD	S		
dynamic_growth	S	S			
employee			S	UD	S
feed	S	S			
inspection_history					
satiety_animal					
type_animal	S	S			

Функції					
	Робочий	Ветеринар	Адміністратор	Завідуючий господарством	login
add_application	EXEC	EXEC			
add_employee				EXEC	
animal_info	EXEC	EXEC	EXEC	EXEC	
employee_info				EXEC	EXEC
feed_animal	EXEC				
get_applications		EXEC		EXEC	
insert_animal	EXEC				
rm_application			EXEC		
weight_animal	EXEC				
get_weighing_history	EXEC				
inspect		EXEC			
change_feed		EXEC			
current_weight	EXEC				
write_off_animals				EXEC	
updateSalary				EXEC	
updateInfoEmployee			EXEC		
rm_employee				EXEC	
update_application			EXEC		



Представлення				
	Робочий	Ветеринар	Адміністратор	Завідуючий господарством
all_aplications			S	
info_animals				S
info_animals_healthy		S		
info_animals_sick		S		
info_animals_hungry	S			
info_animals_not_hungry	S			
info_animals_writeoff				
info_animals_not_writeoff				
all_app_inspection		S	S	
all_app_change_feed			S	
all_app_write_off			S	S

Треба відмітити, що всі функції є захищеними (SECURITY DEFINER). Така функція виконується с правами користувача, що її створив (усі функції були створені дефолтним суперкористувачем). Це дає можливість користувачу без прав доступу до таблиць виконувати всі можливі операції за допомогою лише функцій. Тобто достатньо дати право на виконання функції користувачу.

Створення ролей і надання їм привілеїв відповідно до наведеної вище таблиці здійснюється за допомогою наступних SQL-запитів:

### 1. Адміністратор

```
CREATE ROLE administrator LOGIN
--таблицы
GRANT SELECT ON employee TO administrator;
```

```

GRANT SELECT on clarification_to_app to administrator;
--представления
GRANT SELECT ON all_applications TO administrator;
GRANT SELECT ON all_app_inspection TO administrator;
GRANT SELECT ON all_app_change_feed TO administrator;
GRANT SELECT ON all_app_write_off TO administrator;
--функции
GRANT EXECUTE ON FUNCTION rm_application(numApp int) to
administrator;

GRANT EXECUTE ON FUNCTION animal_info(id integer) to administrator;
GRANT EXECUTE ON FUNCTION updateInfoEmployee(idEmployee int,
loginEmployee varchar(100), passwrд varchar(100), full_name
varchar(100), positionEmployee varchar(100), salaryEmployee int) to
administrator;

GRANT EXECUTE ON FUNCTION update_application(num_application int,
numAnimals int[]) TO administrator;

```

### Лістинг 9.1 - Надання привілеїв адміністратору

## 2. Завідуючий господарством

```

CREATE ROLE head_household LOGIN
-- таблиці
GRANT SELECT,UPDATE,DELETE ON employee TO head_household;
--представления
GRANT SELECT ON all_app_write_off TO head_household;
GRANT SELECT ON info_animals TO head_household;
--функции
GRANT EXECUTE on FUNCTION get_applications(type_application
varchar(100)) to head_household;
GRANT EXECUTE ON FUNCTION animal_info(id integer) to head_household;
GRANT EXECUTE on FUNCTION employee_info(id integer) to
head_household;
GRANT EXECUTE ON FUNCTION rm_employee(idEmployee integer) to
head_household;

```

```
GRANT EXECUTE ON FUNCTION write_off_animals(numAnimals int[],
num_application int, employee_name varchar(100)) to head_household;

GRANT EXECUTE ON function add_employee(log varchar(50), passwd
varchar(50), full_name varchar (100), posit varchar(30), salary int)
to head_household;
```

### Лістинг 9.2 - Надання привілеїв завідувачому господарством

#### 3. Ветеринар

```
CREATE ROLE vet LOGIN

-- таблиці
GRANT SELECT, DELETE on clarification_to_app to vet;
GRANT SELECT on type_animal to vet;
GRANT SELECT on feed to vet;
GRANT SELECT on dynamic_growth to vet;
GRANT SELECT, UPDATE on animal to vet;
GRANT SELECT, DELETE, INSERT on application to vet;

-- представлення
GRANT SELECT on info_animals_healthy to vet;
GRANT SELECT on info_animals_sick to vet;
GRANT SELECT on all_app_inspection to vet;

-- функції
GRANT EXECUTE on FUNCTION animal_info(id integer) to vet;
GRANT EXECUTE on FUNCTION add_application(type_application
varchar(50), animals integer[], employee_name varchar(100)) to vet;
GRANT EXECUTE on FUNCTION get_applications(type_application
varchar(100)) to vet;
GRANT EXECUTE on FUNCTION inspection(employee_name varchar(100),
num_application int, numAnimals int[], statusAnimals varchar[]) to
vet;
GRANT EXECUTE on FUNCTION change_feed(vet varchar(100),
num_application int, numAnimals int[], animalFeed varchar[]) to vet;
```

### Лістинг 9.3 - Надання привілеїв ветеринару

#### 4. Робочий

```

CREATE ROLE worker LOGIN

--таблицы
GRANT SELECT on animal to worker;
GRANT SELECT on type_animal to worker;
GRANT SELECT on feed to worker;
GRANT SELECT on dynamic_growth to worker;

--представления
GRANT SELECT ON info_animals_hungry TO worker
GRANT SELECT ON info_animals_not_hungry TO worker

--функції
GRANT EXECUTE ON FUNCTION animal_info(id integer) to worker;
GRANT EXECUTE ON FUNCTION get_weighing_history(id integer) to
worker;
GRANT EXECUTE ON FUNCTION current_weight(idAnimal integer) to
worker;
GRANT EXECUTE ON FUNCTION feed_animal(worker varchar(100), animals
int[]) to worker;
GRANT EXECUTE ON function weight_animal(employeeID varchar(100),
numAnimal int, weight int) to worker;
GRANT EXECUTE ON FUNCTION add_application(type_application
varchar(50), animals integer[], employee_name varchar(100)) to
worker;
GRANT EXECUTE ON FUNCTION insert_animal(workerName varchar(100),
typeanimal_name varchar(20), feed_name varchar(10), gender
varchar(10), age int, weight int) to worker;

```

#### Лістинг 9.4 - Надання привілеїв робочому

#### 5. Логін

```

CREATE ROLE login LOGIN

GRANT SELECT on employee to login

GRANT EXECUTE on FUNCTION employee_info(id integer) to login;

```

#### Лістинг 9.5 - Надання привілеїв логіну

## 10 ІНСТРУКЦІЯ КОРИСТУВАЧА

В даному розділі розглядається розв’язання задач користувачів за допомогою розробленого інтерфейсу. Як приклад наведено інтерфейс робочого.

Після запуску програми, нас зустрічає вікно входу в систему (Рис. 10.1). Тут користувач має ввести свій логін та пароль у відповідних полях.

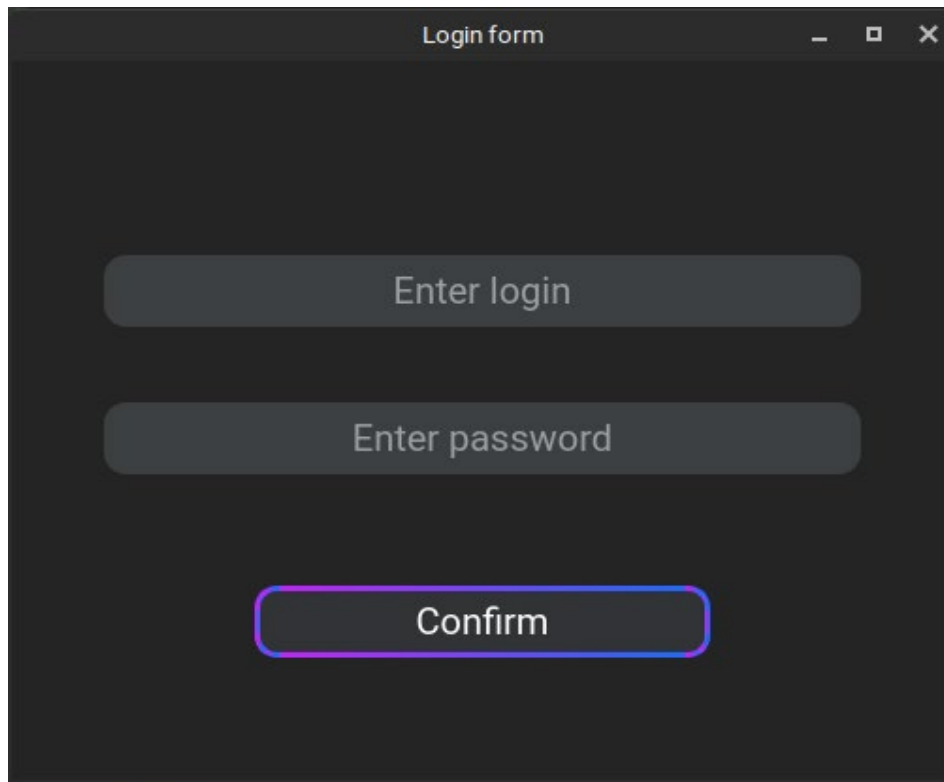


Рисунок 10.1 – Вікно входу

Якщо користувач упізнаний, то з’являється головне вікно програми (Рис. 10.2). Тут знаходиться увесь функціонал доступний для даної ролі, а також коротка інформація про робітника.

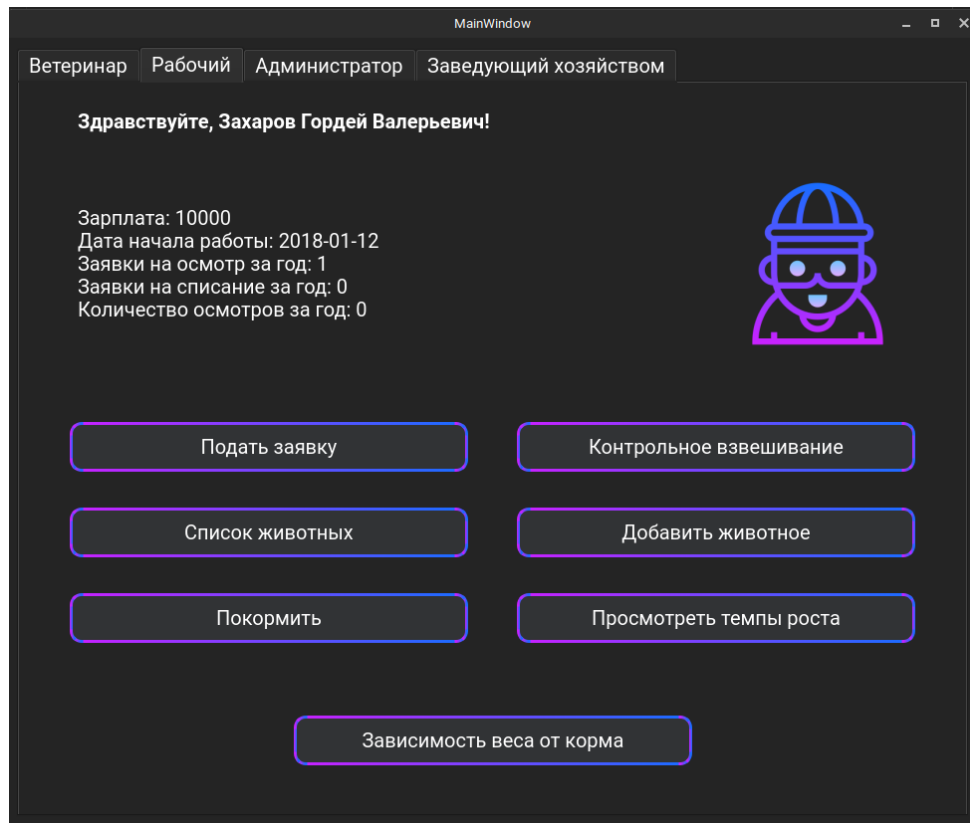


Рисунок 10.2 – Головне вікно

На формі контрольного зважування (Рис. 10.3) при виборі номера тварини відображається коротка інформація про неї, символічні картинки, які відображають основні параметри тварини, а також вагу, яку можливо змінити.

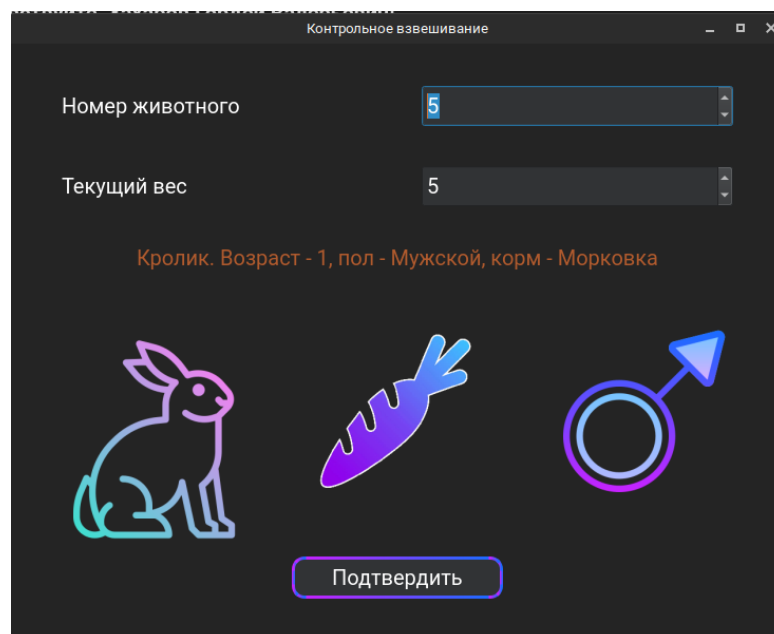


Рисунок 10.3 – Контрольне зважування

При додаванні тварини (Рис. 10.4) є можливість обрати тип тварини, тип корму, стать, вагу та вік у відповідних полях. Також, при зміні параметрів, змінюються і відповідні картинки, які служать для візуалізації обраних параметрів.

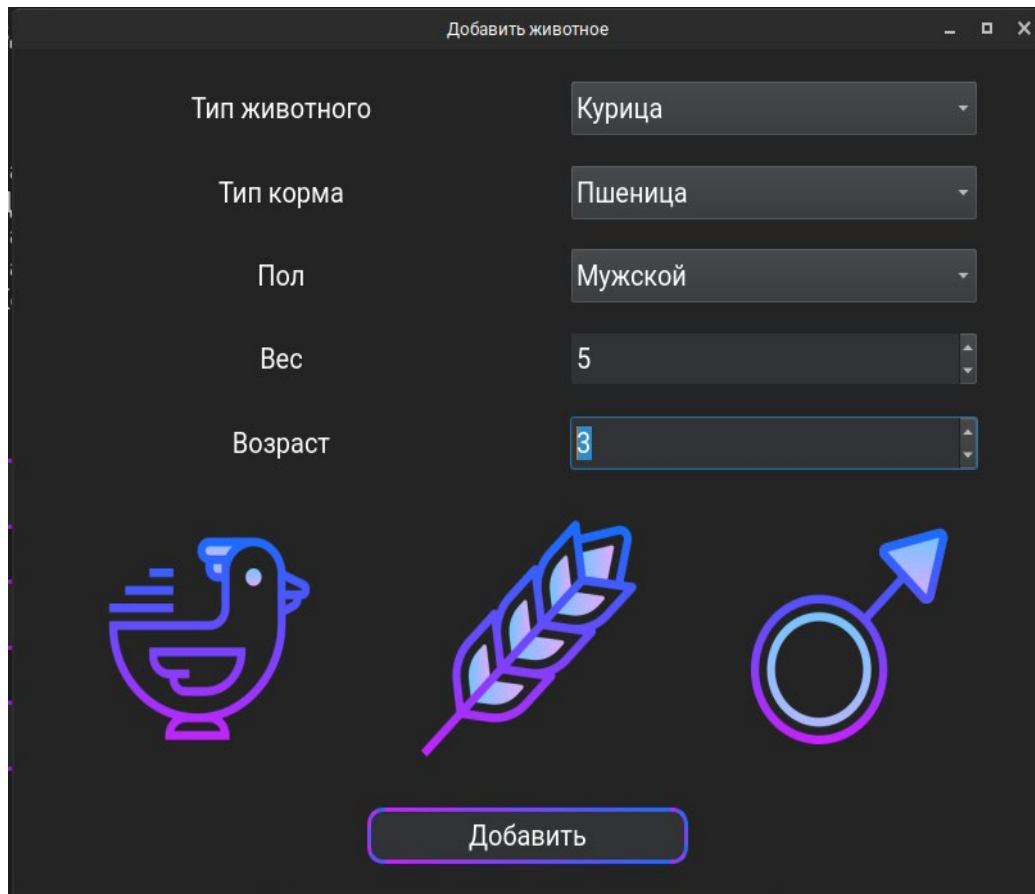


Рисунок 10.4 – Додавання тварини

У ролі робочого є можливість переглядати графіки темпу зростання (Рис. 10.5) та залежності ваги від типу корму (Рис. 10.6). Графік темпу зростання змінюється в залежності від обраного номеру тварини. Усі інші поля форми при цьому стають неактивними. Графік залежності, в свою чергу, змінюється в залежності від обраного типу тварини.

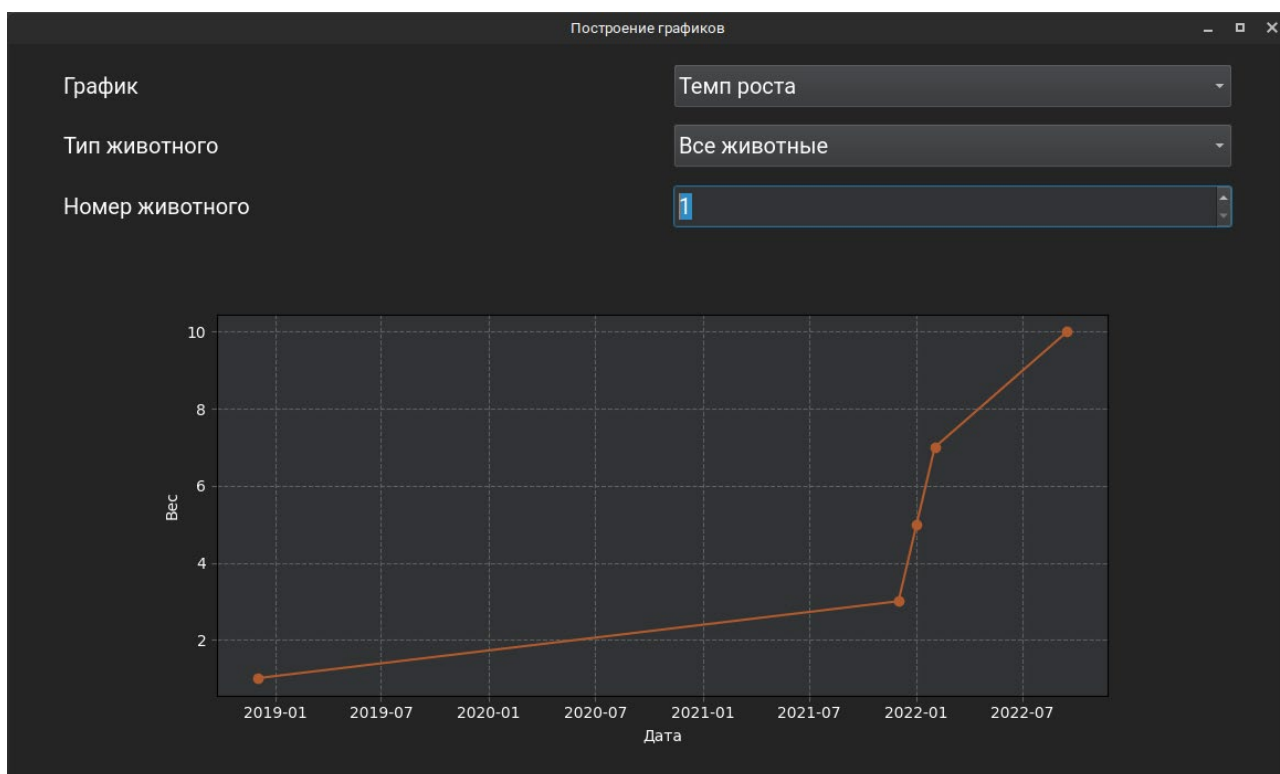


Рисунок 10.5 – Темп зростання

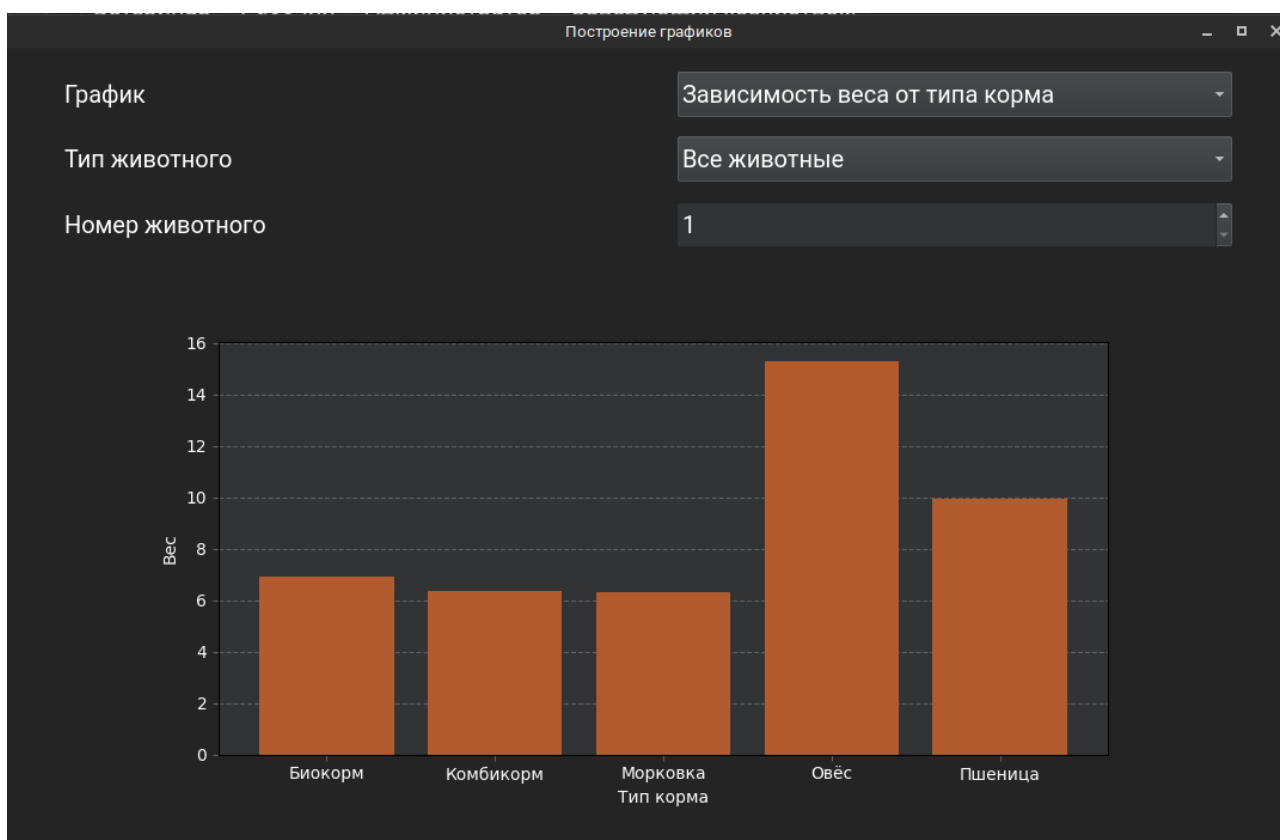


Рисунок 10.6 – Залежність ваги від типу корму



Форма годування (Рис. 10.7) дозволяє додати одразу декілька тварин, що додає зручності у роботі.

Покормить животное

Сотрудник: Захаров Гордей Валерьевич

Номер животного: 14

Добавить Удалить

	Номер	Тип	Пол	Возраст	Корм
1	4	Индейка	Женский	4	Биокорм
2	8	Кролик	Женский	2	Морковка
3	10	Кролик	Женский	3	Морковка
4	12	Страус	Мужской	15	Пшеница
5	14	Страус	Женский	7	Морковка

Подтвердить

Рисунок 10.7 – Залежність ваги від типу корму

У робочого є доступ до списку голодних та ситих тварин (Рис. 10.8). Обрати потрібний список можна змінивши відповідне поле.

Список животных

Голодные животные

	Номер	Тип	Корм	Возраст	Пол
1	2	Индейка	Морковка	4	Мужской
2	3	Индейка	Биокорм	3	Женский
3	4	Индейка	Биокорм	4	Женский
4	5	Кролик	Морковка	1	Мужской
5	6	Кролик	Морковка	2	Мужской
6	7	Кролик	Морковка	3	Мужской
7	8	Кролик	Морковка	2	Женский
8	9	Кролик	Морковка	1	Женский
9	10	Кролик	Морковка	3	Женский
10	11	Страус	Комбикорм	20	Мужской
11	12	Страус	Пшеница	15	Мужской
12	13	Страус	Биокорм	10	Мужской
13	14	Страус	Морковка	7	Женский
14	15	Страус	Овёс	9	Женский
15	16	Страус	Комбикорм	12	Женский
16	17	Курочка	Комбикорм	1	Мужской

Рисунок 10.8 – Залежність ваги від типу корму

Робочий також має можливість залишити заявку (Рис. 10.9). При складанні, є можливість добавляти та видаляти тварин, після чого буде подана нова заявка обраного типу із зазначеним списком тварин.

Добавить заявку

Тип заявки: Заявка на изменение типа корма

Сотрудник: Захаров Гордей Валерьевич

Номер животного: 13

Добавить Удалить

	Номер	Тип	Пол	Возраст	Корм
1	4	Индейка	Женский	4	Биокорм
2	7	Кролик	Мужской	3	Морковка
3	10	Кролик	Женский	3	Морковка
4	13	Страус	Мужской	10	Биокорм

Подтвердить

Рисунок 10.9 – Подання заявки

## ВИСНОВКИ

В даному курсовому проекті виконано аналіз предметної області, в результаті якого визначено користувачів системи та їх задачі, спроектовано і розроблено базу даних на основі СУБД PostgreSQL яка реалізує реляційну модель даних; розроблено з використанням сучасних програмних засобів та бібліотек, таких як Psycopg2.

Інтерфейс клієнтської програми розроблений з урахуванням вимог кожної категорії користувачів і надає необхідний функціонал для вирішення відповідних задач. Доступ до даних з боку різних категорій користувачів розмежований за допомогою механізму ролей і привілеїв. Захист від несанкціонованого доступу реалізований за допомогою використання механізму аутентифікації і авторизації.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Малахов Є.В., Розновець О.І. Проектування інформаційних систем: методичні вказівки до курсового проектування з навчального курсу // Є.В. Малахов, О.І. Розновець / Одеса: Одес. нац. ун-т ім. І.І. Мечникова, 2020.
2. PostgreSQL: The world's most advanced open source database - <http://www.postgresql.org/>
3. PyQt : Краткое руководство - <https://coderlessons.com/tutorials/python-technologies/izuchite-pyqt/pyqt-kratkoe-rukovodstvo>
4. Python для сложных задач: наука о данных и машинное обучение. — СПб.: Питер, 2018. — 576 с.

## ДОДАТОК А. Опис сутностей та їх властивостей

Таблиця А.1 – Опис сутностей та їх властивостей

Властивість	Опис	Обмеження
<b>Employee</b>		
ID_employee	Ідентифікатор співробітника	первинний ключ, не порожнє
Login	Логін співробітника	не порожнє, унікальне
Password	Пароль співробітника	не порожнє
FullName	Повне ім'я співробітника	не порожнє
StartDate	Дата початку роботи	не порожнє, не менше поточної дати
Position	Посада співробітника	Не порожнє, приймає одне із значень («Ветеринар», «Рабочий», «Администратор», «Заведующий хозяйством»)
Status	Являється поміткою, чи працює співробітник	не порожнє, приймає одне із значень («Работает», «Не работает»)
<b>Feed</b>		

## Продовження таблиці А.1

Властивість	Опис	Обмеження
Key_feed	Ідентифікатор типу корму	первинний ключ, не порожнє
Title	Назва типу корму	Не порожнє, приймає одне із значень («Овёс», «Пшеница», «Морковка», «Биокорм», «Комбикорм»)
Calorie_content	Кількість калорій	не порожнє, $\geq 50$ одиниць
<b>Type_animal</b>		
Typeanimal_key	Ідентифікатор типу корму	Первинний ключ, не порожнє
Name_of_type	Назва типу корму	Не порожнє, приймає одне із значень («Хрюша», «Корова», «Курица», «Страус», «Кролик», «Индейка»)
<b>Animal</b>		
Numberanimal	Ідентифікатор тварини	Не порожнє, первинний ключ
ReceiptDate	Дата прибуття	Не порожнє, не менше поточної дати
Key_feed	Ідентифікатор типу корму	Не порожнє, зовнішній ключ
TypeAnimal_key	Ідентифікатор типу тварини	Не порожнє, зовнішній ключ

Продовження таблиці А.1

Властивість	Опис	Обмеження
Age	Вік тварини	Не порожнє, більше 0, >= різниці між поточною датою та датою прибуття
Write_off	Помітка, чи списана тварина	Не може бути менше поточної дати
Gender	Стать тварини	Не порожнє, приймає одне із значень («Мужской», «Женский»)
<b>Satiety_Animal</b>		
Numberanimal	Ідентифікатор тварини	Не порожнє, зовнішній ключ
ID_employee	Ідентифікатор співробітника	Не порожнє, зовнішній ключ
Satiety_key	Ідентифікатор запису про годування	Не порожнє, первинний ключ
Feeding_time	Дата та час годування	Не порожнє
<b>Dynamic_growth</b>		
Dynamic_key	Ідентифікатор запису про зважування	Не порожнє, первинний ключ
Numberanimal	Ідентифікатор тварини	Не порожнє, зовнішній ключ
Date_in	Дата зважування	Не порожнє, не менше поточної дати

## Продовження таблиці А.1

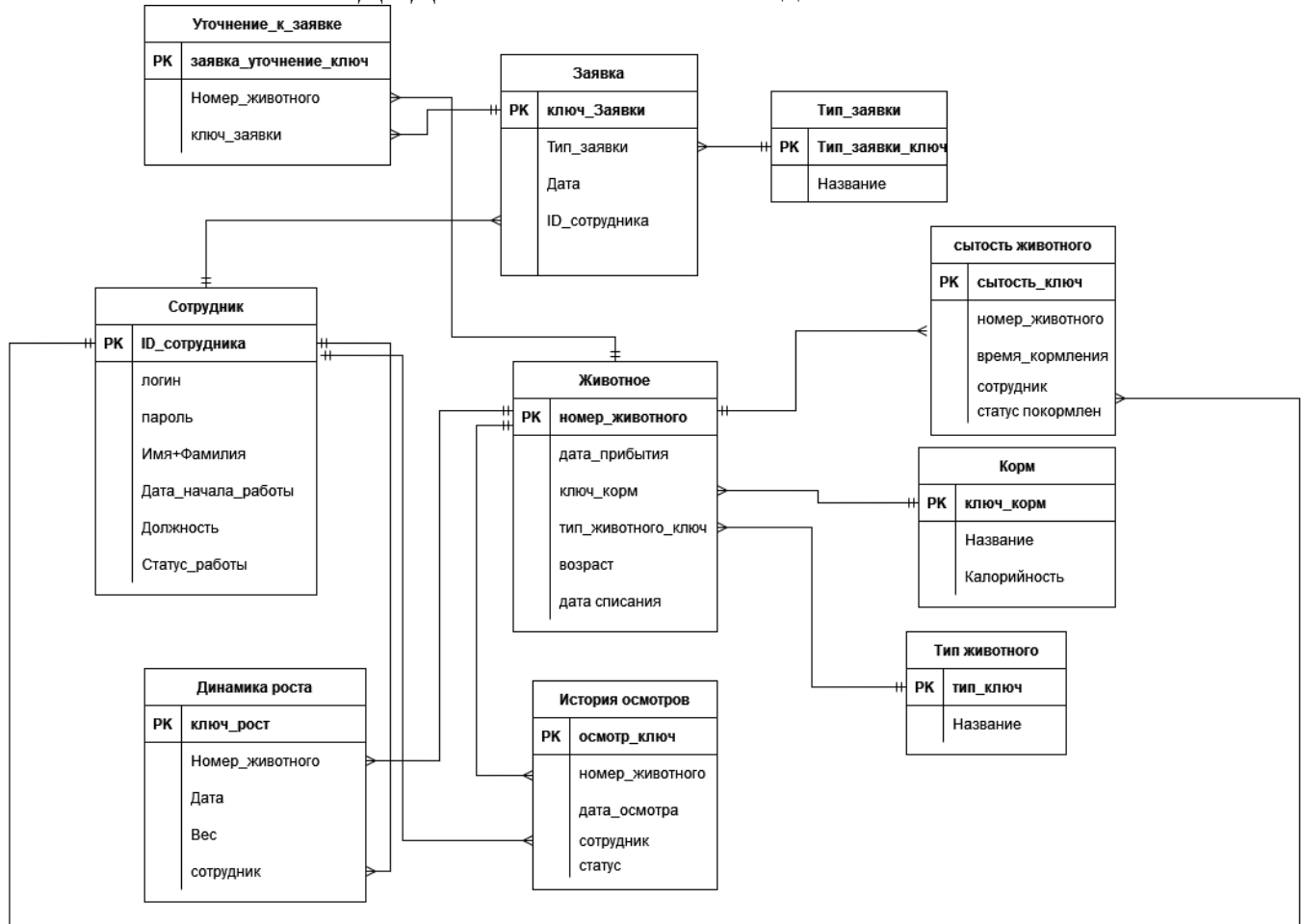
Властивість	Опис	Обмеження
Weight	Вага тварини	Не порожнє, більше 0
ID_employee	Ідентифікатор співробітника	Не порожнє, зовнішній ключ
<b>Inspection_history</b>		
Inspection_key	Ідентифікатор запису про огляд	Не порожнє, первинний ключ
Numberanimal	Ідентифікатор тварини	Не порожнє, зовнішній ключ
ID_employee	Ідентифікатор співробітника	Не порожнє, зовнішній ключ
Date_inspection	Дата огляду	Не порожнє
Status	Статус тварини	Не порожнє, приймає одне із значень («Списанное», «Большое», «Здоровое»)
<b>Application_type</b>		
Application_type_key	Ідентифікатор типу заявки	Не порожнє, первинний ключ
Name_of_app	Назва заявки	Не порожнє, приймає одне із значень («Заявка на списание», «Заявка на осмотр», «Заявка на изменение типа корма»)
<b>Application</b>		



Продовження таблиці А.1

Властивість	Опис	Обмеження
Keyapplication	Ідентифікатор заявки	Не порожнє, первинний ключ
DateOfApp	Дата подачі заявки	Не порожнє, не менше поточної дати
ID_employee	Ідентифікатор співробітника	Не порожнє, зовнішній ключ
Application_type_key	Ідентифікатор типу заявки	Не порожнє, зовнішній ключ
<b>Clarification_to_app</b>		
Clarification_key	Ідентифікатор запису доповнення до заявки	Не порожнє, первинний ключ
Numberanimal	Ідентифікатор тварини	Не порожнє, зовнішній ключ
Keyapplication	Ідентифікатор заявки	Не порожнє, зовнішній ключ

## ДОДАТОК Б. Схема бази даних



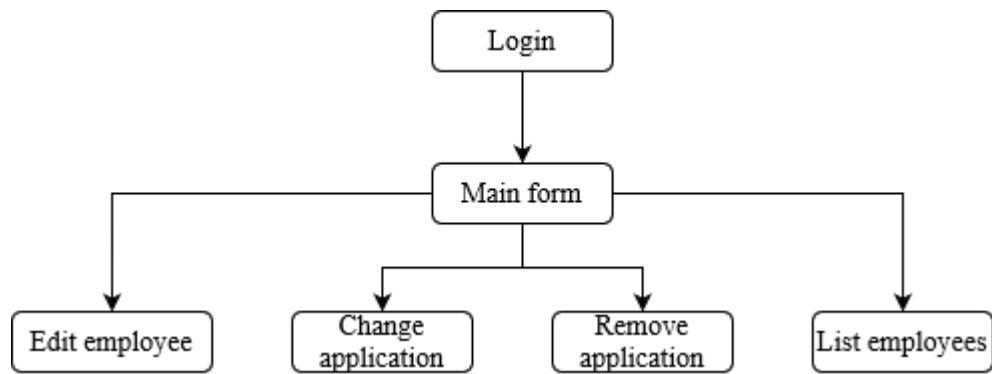
**ДОДАТОК В. Ієрархії елементів системи**

Рисунок В.1 – Ієрархія сторінок застосунку для адміністратора

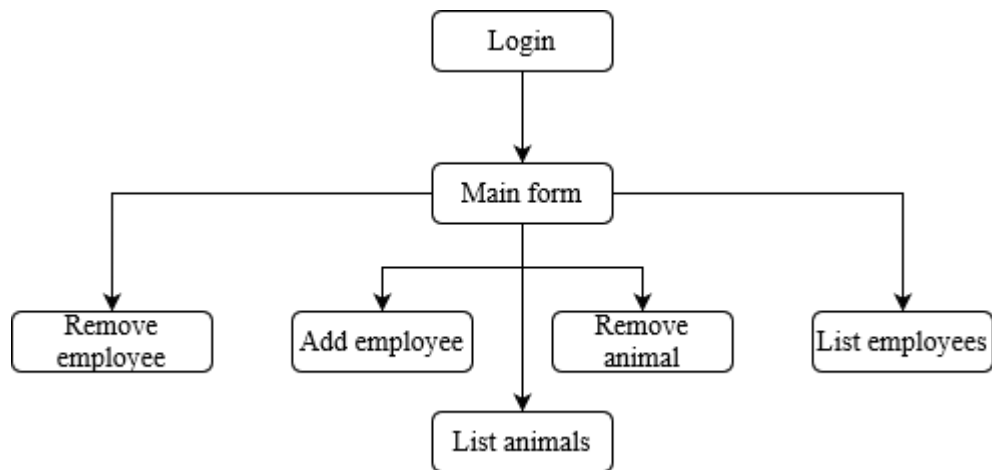


Рисунок В.2 – Ієрархія сторінок застосунку для завідуючого підприємством

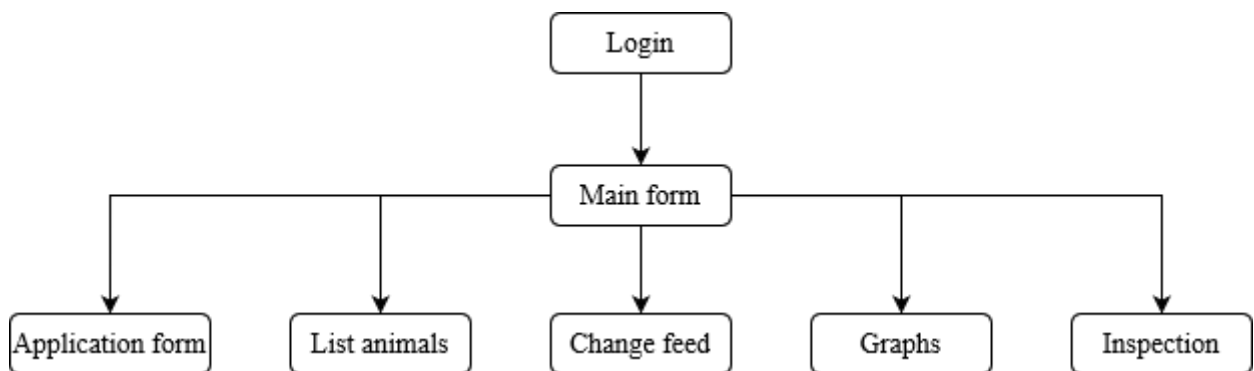


Рисунок В.3 – Ієрархія сторінок застосунку для ветеринара

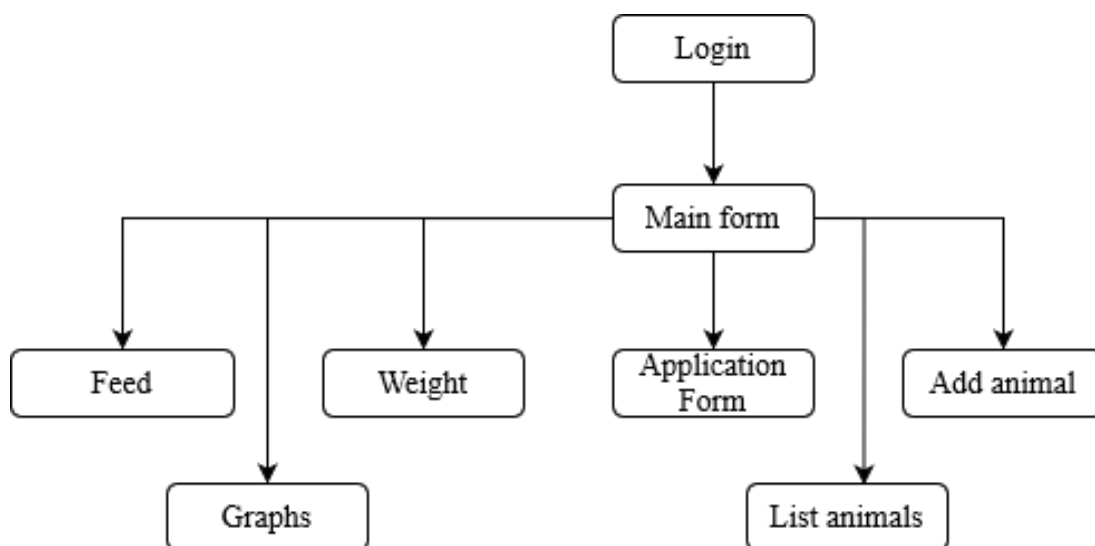


Рисунок В.4 – Ієрархія сторінок застосунку для робочого

## ДОДАТОК Г. Запити на створення таблиць БД

```
CREATE TABLE Employee(
    ID_employee serial not null PRIMARY KEY,
    Login varchar(100) not null UNIQUE,
    Password varchar(100) not null,
    FullName varchar(100) not null,
    StartDate date not null CHECK(StartDate <= CURRENT_DATE),
    Position varchar(40) not null CHECK(Position in ('Ветеринар',
'Рабочий', 'Администратор', 'Заведующий хозяйством')),
    Status varchar(12) not null CHECK(Status in ('Работает', 'Не
работает'))
);
```

### Лістинг Г.1 – Створення таблиці Employee

```
CREATE TABLE Feed(
    Key_Feed serial not null PRIMARY KEY,
    Title varchar(100) not null CHECK(Title in ('Овёс', 'Пшеница',
'Морковка', 'Биокорм', 'Комбикорм')),
    Calorie_content int not null CHECK(Calorie_content > 50)
);
```

### Лістинг Г.2 – Створення таблиці Feed

```
CREATE TABLE Type_Animal(
    TypeAnimal_key serial not null PRIMARY KEY,
    Name_of_type varchar(10) not null CHECK(Name_of_type in
('Хрюша', 'Корова', 'Курица', 'Страус', 'Кролик', 'Индейка'))
);
```

### Лістинг Г.3 – Створення таблиці Type\_Animal

```
CREATE TABLE Animal(
    NumberAnimal serial not null PRIMARY KEY,
    ReceiptDate date not null CHECK(ReceiptDate <= CURRENT_DATE),
    Key_Feed serial not null,
    TypeAnimal_key serial not null,
```

```

    Age int not null CHECK(Age > 0 and Age >= ReceiptDate-
CURRENT_DATE),
    Write_off date CHECK(Write_off <= CURRENT_DATE),
    Gender varchar(10) not null CHECK(Gender in
('Мужской', 'Женский')),

    FOREIGN KEY(Key_Feed) REFERENCES Feed(Key_Feed)
    ON UPDATE CASCADE ON DELETE RESTRICT,

    FOREIGN KEY(TypeAnimal_key) REFERENCES
Type_Animal(TypeAnimal_key)
    on update cascade on delete restrict
);

```

#### Лістинг Г.4 – Створення таблиці Animal

```

CREATE TABLE Satiety_Animal(
    NumberAnimal serial not null,
    ID_employee serial not null,
    Satiety_key serial not null PRIMARY KEY,
    Feeding_time timestamp not null,

    FOREIGN KEY(ID_employee) REFERENCES Employee(ID_employee)
    on update cascade on delete restrict,

    FOREIGN KEY(NumberAnimal) REFERENCES Animal(NumberAnimal)
    on update cascade on delete restrict
);

```

#### Лістинг Г.5 – Створення таблиці Animal

```

CREATE TABLE Dynamic_growth(
    Dynamic_key serial not null PRIMARY KEY,
    NumberAnimal serial not null,
    Date_in date not null CHECK(Date_in <= CURRENT_DATE),
    Weight int not null CHECK(Weight > 0),
    ID_employee serial not null,

```

```

FOREIGN KEY(ID_employee) REFERENCES Employee(ID_employee)
on update cascade on delete restrict,

FOREIGN KEY(NumberAnimal) REFERENCES Animal(NumberAnimal)
on update cascade on delete restrict
);

```

#### Лістинг Г.6 – Створення таблиці Dynamic\_growth

```

CREATE TABLE Inspection_history(
    Inspection_key serial not null PRIMARY KEY,
    NumberAnimal serial not null,
    ID_employee serial not null,
    Date_inspection date not null,
    Status varchar(10) not null CHECK(Status in ('Списанное',
'Бо́льное', 'Здоровое'))),

FOREIGN KEY(NumberAnimal) REFERENCES Animal(NumberAnimal)
on update cascade on delete restrict,

FOREIGN KEY(ID_employee) REFERENCES Employee(ID_employee)
on update cascade on delete restrict
);

```

#### Лістинг Г.7 – Створення таблиці Inspection\_history

```

CREATE TABLE Application_type(
    Application_type_key serial not null PRIMARY KEY,
    Name_of_app varchar(50) not null CHECK( Name_of_app in ('Заявка
на списание', 'Заявка на осмотр', 'Заявка на изменение типа корма'))
);

```

#### Лістинг Г.8 – Створення таблиці Application\_type

```

CREATE TABLE Application(

```

```

KeyApplication serial not null PRIMARY KEY,
DateOfApp timestamp not null CHECK(DateOfApp <= CURRENT_DATE),
ID_employee serial not null,
Application_type_key serial not null,

FOREIGN          KEY(Application_type_key)          REFERENCES
Application_type(Application_type_key)
on update cascade on delete restrict,

FOREIGN KEY(ID_employee) REFERENCES Employee(ID_employee)
on update cascade on delete restrict
);

```

### Лістинг Г.9 – Створення таблиці Application

```

CREATE TABLE Clarification_to_app(
    Clarification_key serial not null PRIMARY KEY,
    NumberAnimal serial not null,
    KeyApplication serial not null,

    FOREIGN KEY(NumberAnimal) REFERENCES Animal(NumberAnimal)
on update cascade on delete restrict,

FOREIGN          KEY(KeyApplication)          REFERENCES
Application(KeyApplication)
on update cascade on delete restrict
);

```

### Лістинг Г.10 – Створення таблиці Clarification\_to\_app



## ДОДАТОК Д. Запити на створення тригерів

```

CREATE OR REPLACE FUNCTION hash_password()
RETURNS trigger AS
$$
    BEGIN
        IF (TG_TABLE_NAME = 'employee')
        THEN
            IF (TG_OP = 'INSERT' OR NEW.password NOT LIKE
OLD.password)
            THEN
                NEW.password = encode(digest(NEW.password,
'sha256'), 'hex');
            END IF;
        END IF;
        RETURN NEW;
    END;
$$
LANGUAGE 'plpgsql';

```

### Лістинг Д.1 – Створення тригеру hash\_password()

```

CREATE FUNCTION satiety_func()
RETURNS TRIGGER
AS $$
BEGIN
    IF (SELECT write_off FROM animal WHERE animal.numberanimal
= NEW.numberanimal) IS NOT NULL THEN
        RAISE EXCEPTION 'Животное не может быть списано !';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

```

CREATE TRIGGER satiety_trg
BEFORE INSERT ON satiety_animal
FOR EACH ROW
EXECUTE PROCEDURE satiety_func();

```

### Лістинг Д.2 – Створення тригеру satiety\_func()

## ДОДАТОК Е. Запити на створення функцій

```

CREATE OR REPLACE FUNCTION add_application(type_application
varchar(50), animals integer[], employee_name varchar(100)) RETURNS
void
AS $$
DECLARE
    number_app int;
    employeeID int;
    appID int;
    numAnimal int := 0; -- счётчик в цикле
begin
-- проверки

    -- проверка на существование типа заявки
    IF lower(type_application) IN (SELECT lower(name_of_app) FROM
application_type) THEN
        number_app := (SELECT application_type_key FROM
application_type WHERE lower(name_of_app) =
lower(type_application));
        --RAISE NOTICE 'number_app %', number_app;
    ELSE
        RAISE EXCEPTION 'Не существует такого типа заявки !';
    END IF;

    -- проверка на существование сотрудника
    IF lower(employee_name) IN (SELECT lower(fullname) FROM
employee) THEN
        employeeID:= (SELECT id_employee FROM employee WHERE
lower(fullname) = lower(employee_name));
        --RAISE NOTICE 'employee ID = %', employeeID;
    ELSE
        RAISE EXCEPTION 'Не существует такого сотрудника !';
    END IF;

    -- проверка должности сотрудника
    IF number_app = 1 AND (SELECT position FROM employee WHERE
id_employee = employeeID) != 'Ветеринар' THEN
        RAISE EXCEPTION 'Заявку на списание может подать только
ветеринар !';
    ELSEIF number_app = 2 AND (SELECT position FROM employee WHERE
id_employee = employeeID) != 'Рабочий' THEN
        RAISE EXCEPTION 'Заявку на осмотр может подать только
рабочий !';
    ELSEIF number_app = 3 AND (SELECT position FROM employee WHERE
id_employee = employeeID) != 'Рабочий' THEN
        RAISE EXCEPTION 'Заявку на изменение типа корма может подать
только рабочий !';
    END IF;

```

```

-- проверка существует(списано) ли каждое животное
FOREACH numAnimal IN ARRAY animals LOOP
    IF (SELECT gender FROM animal WHERE numberanimal =
numAnimal) IS NULL OR (SELECT write_off FROM animal WHERE
numberanimal = numAnimal) IS NOT NULL THEN
        RAISE EXCEPTION 'Животного с номером % не существует или
оно списано !', numAnimal;
    END IF;
END LOOP;

-- создаём заявку
INSERT INTO Application(DateOfApp, ID_employee,
Application_type_key) VALUES (CURRENT_DATE, employeeID, number_app)
RETURNING keyapplication INTO appID;
numAnimal := 0;
FOREACH numAnimal IN ARRAY animals LOOP
    INSERT INTO clarification_to_app(KeyApplication,
numberanimal) VALUES (appID, numAnimal);
END LOOP;

END;
$$ language plpgsql
SECURITY DEFINER;

```

### Лістинг Е.1 – Створення функції add\_application

```

CREATE OR REPLACE FUNCTION insert_animal(workerName varchar(100),
typeanimal_name varchar(20), feed_name varchar(10), gender
varchar(10), age int, weight int) RETURNS void
AS $$
DECLARE
    number_an int;
    worker int;
    vet int;

    type_animal_int int;
    feed_int int;

    substr text;
    type_list text = '';
    feed_list text = '';

begin
    -- записуємо список допустимих типів тварин і корма в
    -- відповідуючі змінні
    -- Нужно для отображения в ошибке

    for substr in (select name_of_type FROM type_animal) loop
        type_list := type_list || substr || ' ';
    end loop;

    for substr in (SELECT title FROM feed) loop

```

```

        feed_list := feed_list || substr || ' ';
    end loop;

    -- Проверка каждого переданного параметра
    IF lower(workerName) IN (SELECT lower(fullname) FROM
employee) THEN
        worker:= (SELECT id_employee FROM employee WHERE
lower(fullname) = lower(workerName));
    ELSE
        RAISE EXCEPTION 'Не существует такого сотрудника !';
    END IF;
    IF lower(typeanimal_name) NOT IN (SELECT
lower(name_of_type) FROM type_animal)
    THEN RAISE EXCEPTION 'Не найдено такого типа
животного. Допустимые типы: ', type_list;
    ELSEIF lower(feed_name) NOT IN (SELECT lower(title) FROM
feed)
    THEN RAISE EXCEPTION 'Неизвестный тип корма.
Допустимые значения: ', feed_list;
    ELSEIF age < 0
    THEN RAISE EXCEPTION 'Возраст не может быть
отрицательным !';
    ELSEIF lower(gender) NOT IN ('мужской', 'женский')
    THEN RAISE EXCEPTION 'Неизвестный пол животного.
Допустимые значения: Мужской, Женский';
    ELSEIF weight < 0
    THEN RAISE EXCEPTION 'Вес не может быть отрицательным
!';
    END IF;

    -- переведем в тип животного и корма в соответствующее
число
    type_animal_int := (SELECT typeanimal_key FROM type_animal
WHERE name_of_type = typeanimal_name);
    feed_int := (SELECT key_feed FROM feed WHERE title =
feed_name);

    -- Выбираем случайного ветеринара
    vet := (SELECT id_employee from (select * from employee
where position = 'Ветеринар' and status = 'Работает') as employee
ORDER BY RANDOM()
LIMIT 1);

    RAISE NOTICE 'Worker - ', worker ;
    RAISE NOTICE 'Vet - ', vet;

    INSERT INTO animal(ReceiptDate, Key_Feed, TypeAnimal_key,
Age, Gender) VALUES (CURRENT_DATE, feed_int, type_animal_int, age,
gender) RETURNING numberanimal INTO number_an;
    INSERT INTO Dynamic_growth(NumberAnimal, Date_in, Weight,
ID_employee) VALUES (number_an, CURRENT_DATE, weight, worker);

```

```

        INSERT INTO Satiety_Animal(NumberAnimal, ID_employee,
Feeding_time) VALUES (number_an, worker , NOW());
        INSERT INTO Inspection_history(NumberAnimal, ID_employee,
Date_inspection, Status) VALUES (number_an, vet, CURRENT_DATE,
'Здоровое');

        RAISE NOTICE 'Number new animal %', number_an;
END;
$$ LANGUAGE plpgsql
SECURITY DEFINER;

```

## Лістинг Е.2 – Створення функції insert\_animal

```

create or replace function add_employee(log varchar(50), passwrд
varchar(50), full_name varchar (100), posit varchar(30), salary int)
returns void
as $$
declare
    employee_list text = '';
    substr text;
    number_employee int;
begin
    for substr in (select distinct position from employee)
    loop
        employee_list := employee_list || substr || '; ';
    end loop;

    if log in (select login from employee)
        then raise exception 'Такой логин уже существует !';
    elseif lower(posit) not in (select distinct lower(position)
from employee)
        then raise exception 'Не существует такой должности.
Доступные варианты: %', employee_list;
    elseif salary <= 0
        then raise exception 'Зарплата не может быть меньше или
равна 0 !';
    end if;

    insert into employee(login, password, fullname, startdate,
position, status, salary) values (log, passwrд, full_name,
CURRENT_DATE, posit, 'Работает', salary) returning id_employee into
number_employee ;
    raise notice 'ID new employee - %', number_employee;
end;
$$ language plpgsql
SECURITY DEFINER;

```

## Лістинг Е.3 – Створення функції add\_employee

```

create or replace function feed_animal(worker varchar(100), animals
int[])
returns void
as $$
declare
    animalID int = 0;
    workerID int = 0;
begin
    IF lower(worker) IN (SELECT lower(fullname) FROM employee WHERE
status = 'Работает') AND (SELECT position FROM employee WHERE
lower(fullname) = lower(worker)) = 'Рабочий' THEN
        workerID:= (SELECT id_employee FROM employee WHERE
lower(fullname) = lower(worker));
    ELSE
        RAISE EXCEPTION 'Не существует такого действующего
рабочего !';
    END IF;

    -- проверка существует ли каждое животное
    FOREACH animalID IN ARRAY animals LOOP
        IF (SELECT gender FROM animal WHERE numberanimal = animalID)
IS NULL THEN
            RAISE EXCEPTION 'Животного с номером % не существует !',
animalID;
        END IF;
    END LOOP;

    FOREACH animalID IN ARRAY animals LOOP
        INSERT INTO Satiety_Animal(NumberAnimal, ID_employee,
Feeding_time) VALUES (animalID, workerID , NOW());
    END LOOP;
end;
$$ language plpgsql
SECURITY DEFINER;

```

#### Лістинг Е.4 – Створення функції feed\_animal

```

create or replace function weight_animal(employeeID varchar(100),
numAnimal int, weight int)
returns void
as $$
declare
    workerID int;
begin
    IF lower(employeeID) IN (SELECT lower(fullname) FROM employee
WHERE status = 'Работает') AND (SELECT position FROM employee WHERE
lower(fullname) = lower(employeeID)) = 'Рабочий' THEN
        workerID:= (SELECT id_employee FROM employee WHERE
lower(fullname) = lower(employeeID));
    ELSE
        RAISE EXCEPTION 'Не существует такого действующего рабочего
!';

```

```

END IF;

IF weight <= 0 THEN
    RAISE EXCEPTION 'Вес не может быть меньше 0 !';
END IF;

IF (SELECT gender FROM animal WHERE numberanimal = numAnimal)
IS NULL OR (SELECT write_off FROM animal WHERE numberanimal =
numAnimal) IS NOT NULL THEN
    RAISE EXCEPTION 'Животного с номером % не существует или оно
списано !', numAnimal;
ELSE
    INSERT INTO Dynamic_growth(NumberAnimal, Date_in, Weight,
ID_employee) VALUES (numAnimal, CURRENT_DATE, weight, workerID);
END IF;
end;
$$ language plpgsql
SECURITY DEFINER;

```

#### Лістинг E.5 – Створення функції weight\_animal

```

create or replace function rm_application(numApp int)
returns void
as $$
declare
    appID int;
begin
    IF (select id_employee from application where keyapplication =
numApp) IS NOT NULL THEN
        DELETE FROM clarification_to_app WHERE keyapplication =
numApp;
        DELETE FROM application WHERE keyapplication = numApp;
    ELSE
        RAISE EXCEPTION 'Неверный номер заявки !';
    END IF;
end;
$$ language plpgsql
SECURITY DEFINER;

```

#### Лістинг E.6 – Створення функції rm\_application

```

CREATE OR REPLACE FUNCTION animal_info(id integer)
RETURNS table("Номер" int, "Тип" varchar(100), "Пол" varchar(100),
"Возраст" int, "Корм" varchar(100))
AS $$
begin
    IF (SELECT gender FROM animal WHERE numberanimal = id) IS NULL OR
(SELECT write_off FROM animal WHERE numberanimal = id) IS NOT NULL
THEN
        RAISE EXCEPTION 'Животного с номером % не существует или оно
списано !', id;
    END IF;

```

```

RETURN QUERY SELECT numberanimal, name_of_type, gender, age,
title
                from animal
                JOIN type_animal ON animal.typeanimal_key =
type_animal.typeanimal_key
                JOIN feed ON animal.key_feed = feed.key_feed
                where numberanimal = id
                order by numberanimal;

end;
$$ LANGUAGE plpgsql
SECURITY DEFINER;
```

### Лістинг Е.7 – Створення функції animal\_info

```

CREATE OR REPLACE FUNCTION get_applications(type_application
varchar(100))
RETURNS table(application varchar(100), numApplication int)
AS $$
DECLARE
    number_app int;
begin
    -- проверка на существование типа заявки
    IF lower(type_application) IN (SELECT lower(name_of_app) FROM
application_type) THEN
        number_app := (SELECT application_type_key FROM
application_type WHERE lower(name_of_app) =
lower(type_application));
        --RAISE NOTICE 'number_app %', number_app;
    ELSE
        RAISE EXCEPTION 'Не существует такого типа заявки !';
    END IF;

    RETURN QUERY select name_of_app, keyapplication
                    FROM application
                    JOIN application_type ON
application.application_type_key =
application_type.application_type_key
                    WHERE application.application_type_key =
number_app;

end;
$$ LANGUAGE plpgsql
SECURITY DEFINER;
```

### Лістинг Е.8 – Створення функції get\_applications



```

CREATE OR REPLACE FUNCTION employee_info(id integer)
RETURNS table(name varchar(100), "Зарплата" int, "Начало работы"
date, "Должность" varchar(100), "заявки на осмотр" bigint, "заявки
на списание" bigint, "количество осмотров" bigint)
AS $$
BEGIN

    IF id NOT IN (SELECT id_employee FROM employee)
    THEN RAISE EXCEPTION 'not found!';
    END IF;
    RETURN QUERY SELECT  employee.fullname,
                        employee.salary,
                        employee.startdate,
                        employee.position,

                        (SELECT COUNT(*) FROM application WHERE
application.id_employee = id
                        AND dateofapp > CURRENT_DATE - interval'1 year'
AND application_type_key = 2) as "заявки на осмотр",

                        (SELECT COUNT(*) FROM application WHERE
application.id_employee = id
                        AND dateofapp > CURRENT_DATE - interval'1 year'
AND application_type_key = 1) as "заявки на списание",

                        (SELECT COUNT(*) FROM inspection_history WHERE
inspection_history.id_employee = id) as "количество осмотров"
FROM employee
WHERE id_employee = id;
END;
$$ LANGUAGE plpgsql
SECURITY DEFINER;

```

### Лістинг Е.9 – Створення функції employee\_info

```

CREATE OR REPLACE FUNCTION get_weighing_history(id integer)
RETURNS table(date_in date, weight integer)
AS $$
BEGIN
    IF id NOT IN (SELECT numberanimal FROM animal) THEN
        RAISE EXCEPTION 'Животного с номером % не существует !', id;
    END IF;
    RETURN QUERY SELECT dynamic_growth.date_in,
dynamic_growth.weight FROM dynamic_growth
WHERE numberanimal = id;
END;
$$ LANGUAGE plpgsql
SECURITY DEFINER;

```

### Лістинг Е.10 – Створення функції get\_weighing\_history

```

CREATE OR REPLACE FUNCTION inspection(employee_name varchar(100),
num_application int, numAnimals int[], feedForAnimals varchar[])
RETURNS void
AS $$
DECLARE
    write_off_animals int[];
    index_write_off int := 0;
    employeeID int;
BEGIN
    -- проверка на существование сотрудника
    IF lower(employee_name) IN (SELECT lower(fullname) FROM
employee) THEN
        employeeID:= (SELECT id_employee FROM employee WHERE
lower(fullname) = lower(employee_name));
    ELSE
        RAISE EXCEPTION 'Не существует такого сотрудника !';
    END IF;

    -- проверка, существует ли такая заявка
    IF num_application NOT IN (SELECT keyapplication FROM
all_app_inspection) THEN
        RAISE EXCEPTION 'Не существует такой заявки на осмотр !';
    END IF;

    -- проверка, совпадает ли длина массивов
    IF array_length(numAnimals, 1) !=
array_length(statusAnimals,1) THEN
        RAISE EXCEPTION 'Не совпадает количество животных и их
кормов !';
    END IF;

    -- совпадает ли количество животных в заявке с количеством
животных поступающих в функцию
    IF array_length(numAnimals, 1) != (SELECT COUNT(numberanimal)
FROM all_app_inspection WHERE keyapplication = num_application) THEN
        RAISE EXCEPTION 'Не совпадает количество животных в заявке
!';
    END IF;

    --проверка каждого животного на нахождение в заявке
    FOR i IN 0..array_length(numAnimals, 1) LOOP
        IF numAnimals[i] NOT IN (SELECT numberanimal FROM
all_app_inspection WHERE keyapplication=num_application) THEN
            RAISE EXCEPTION 'Животное номер % не указано в заявке !',
i;
        END IF;
    END LOOP;

    -- правильно ли указаны состояния животных
    FOR i IN 0..array_length(statusAnimals, 1) LOOP
        IF statusAnimals[i] NOT IN ('Болезное', 'Здоровое', 'В
заявку на списание') THEN

```

```

        RAISE EXCEPTION 'Неправильно указано состояние одного из
животных: %', statusAnimals[i];
    END IF;
END LOOP;

-- заполняем массив списанных животных
FOR i IN 1..array_length(statusAnimals,1) LOOP
    IF statusAnimals[i] IN ('В заявку на списание') THEN
        write_off_animals[index_write_off] = numAnimals[i];
        index_write_off := index_write_off + 1;
    END IF;
END LOOP;

--если массив списанных животных не пустой, составляем заявку
на списание
IF array_length(write_off_animals,1) IS NOT NULL THEN
    PERFORM add_application('Заявка на списание',
write_off_animals, employee_name);
END IF;

-- проверяем наличие не списанных животных и обновляем их
состояния
FOR i IN 0..array_length(numAnimals,1) LOOP
    IF statusAnimals[i] NOT IN ('В заявку на списание') THEN
        INSERT INTO Inspection_history(NumberAnimal,
ID_employee, Date_inspection, Status) VALUES (numAnimals[i],
employeeID, CURRENT_DATE, statusAnimals[i]);
    END IF;
END LOOP;

--удаляем заявку
PERFORM rm_application(num_application);

END;
$$ LANGUAGE plpgsql
SECURITY DEFINER

```

### Лістинг Е.11 – Створення функції inspection

```

CREATE OR REPLACE FUNCTION change_feed(vet varchar(100),
num_application int, numAnimals int[], animalFeed varchar[])
RETURNS void
AS $$
DECLARE
    employeeID int;
    feedID int[];
BEGIN
    -- проверка на существование сотрудника
    IF lower(vet) IN (SELECT lower(fullname) FROM employee WHERE
status = 'Работает') AND (SELECT position FROM employee WHERE
lower(fullname) = lower(vet)) = 'Ветеринар' THEN

```

```

        employeeID := (SELECT id_employee FROM employee WHERE
lower(fullname) = lower(vet));
    ELSE
        RAISE EXCEPTION 'Не существует такого действующего
ветеринара !';
    END IF;

    -- проверка, существует ли такая заявка
    IF num_application NOT IN (SELECT keyapplication FROM
all_app_change_feed) THEN
        RAISE EXCEPTION 'Не существует такой заявки на изменение
типа корма !';
    END IF;

    -- проверка, совпадает ли длина массивов
    IF array_length(numAnimals, 1) != array_length(animalFeed,1)
THEN
        RAISE EXCEPTION 'Не совпадает количество животных и их
кормов !';
    END IF;

    -- совпадает ли количество животных в заявке с количеством
животных поступающих в функцию
    IF array_length(numAnimals, 1) != (SELECT COUNT(numberanimal)
FROM all_app_change_feed WHERE keyapplication = num_application)
THEN
        RAISE EXCEPTION 'Не совпадает количество животных в заявке
!';
    END IF;

    --проверка каждого животного на нахождение в заявке
    FOR i IN 0..array_length(numAnimals, 1) LOOP
        IF numAnimals[i] NOT IN (SELECT numberanimal FROM
all_app_change_feed WHERE keyapplication=num_application) THEN
            RAISE EXCEPTION 'Животное номер % не указано в заявке !',
numAnimals[i];
        END IF;
    END LOOP;

    -- правильно ли указаны корма животных и запоминаем ключи
    FOR i IN 1..array_length(animalFeed, 1) LOOP
        IF animalFeed[i] NOT IN (SELECT title FROM feed) THEN
            RAISE EXCEPTION 'Неправильно указан корм одного из
животных: %', animalFeed[i];
        END IF;
        feedID[i] = (SELECT key_feed FROM feed WHERE title =
animalFeed[i]);
    END LOOP;

    RAISE INFO 'ID`s = %', feedID;

    -- обновляем корма

```

```

FOR i IN 1..array_length(feedID,1) LOOP
    UPDATE animal
    SET key_feed = feedID[i]
    WHERE numberanimal = numAnimals[i];
END LOOP;

--удаляем заявку
PERFORM rm_application(num_application);
END;
$$ language plpgsql
SECURITY DEFINER;

```

### Лістинг Е.12 – Створення функції change\_feed

```

CREATE OR REPLACE FUNCTION current_weight(idAnimal int)
RETURNS int
AS $$
BEGIN
    IF idAnimal NOT IN (SELECT numberanimal FROM animal) OR idAnimal
    IN (SELECT numberanimal FROM animal WHERE write_off IS NOT NULL)
    THEN
        RAISE EXCEPTION 'Животного с номером % не существует или оно
        списано!', idAnimal;
    END IF;
    RETURN (SELECT weight FROM get_weighing_history(idAnimal) WHERE
    date_in IN (SELECT MAX(date_in) FROM
    get_weighing_history(idAnimal)));
END;
$$ LANGUAGE plpgsql
SECURITY DEFINER;

```

### Лістинг Е.13 – Створення функції current\_weight

```

CREATE OR REPLACE FUNCTION updateSalary(idEmployee int,
salaryEmployee int)
RETURNS void
AS $$
BEGIN
    IF idEmployee NOT IN (SELECT id_employee FROM employee) OR
    (SELECT status FROM employee WHERE id_employee = idEmployee) = 'Не
    работает' THEN
        RAISE EXCEPTION 'Сотрудника с номером % не существует
    или он не работает!', idEmployee;
    END IF;
    IF salaryEmployee == 0 THEN
        RAISE EXCEPTION 'Зарплата не может быть равно 0 !';
    END IF;

    UPDATE employee SET salary = salaryEmployee WHERE id_employee
    = idEmployee;
END;
$$ LANGUAGE plpgsql

```

SECURITY DEFINER;

### Лістинг Е.14 – Створення функції updateSalary

```
CREATE OR REPLACE FUNCTION updateInfoEmployee(idEmployee int,
loginEmployee varchar(100), passwrд varchar(100), full_name
varchar(100), positionEmployee varchar(100), salaryEmployee int)
RETURNS void
AS $$
BEGIN
    IF idEmployee NOT IN (SELECT id_employee FROM employee) OR
(SELECT status FROM employee WHERE id_employee = idEmployee) = 'Не
работает' THEN
        RAISE EXCEPTION 'Сотрудника с номером % не существует или
он не работает!', idEmployee;
    END IF;

    IF loginEmployee NOT IN (SELECT login FROM employee WHERE
id_employee = idEmployee) THEN
        IF loginEmployee IN (SELECT login FROM employee) THEN
            RAISE EXCEPTION 'Такой логин уже существует у другого
сотрудника!';
        END IF;
    END IF;

    IF salaryEmployee <= 0 THEN
        RAISE EXCEPTION 'Зарплата не может быть меньше или равна
нулю !';
    END IF;

    IF positionEmployee NOT IN ('Ветеринар', 'Рабочий',
'Администратор', 'Заведующий хозяйством') THEN
        RAISE EXCEPTION 'Не существует такой должности !';
    END IF;

    UPDATE employee SET
        login = loginEmployee,
        password = passwrд,
        fullname = full_name,
        salary = salaryEmployee
    WHERE id_employee = idEmployee;

END;
$$ LANGUAGE plpgsql
SECURITY DEFINER;
```

### Лістинг Е.15 – Створення функції updateInfoEmployee

```
CREATE OR REPLACE FUNCTION rm_employee(idEmployee int)
RETURNS void
AS $$
DECLARE
    positionEmployee varchar(100);
BEGIN
```

```

        IF idEmployee NOT IN (SELECT id_employee FROM employee) OR
        (SELECT status FROM employee WHERE id_employee = idEmployee) = 'Не
        работает' THEN
            RAISE EXCEPTION 'Сотрудника с номером % не существует или
            он не работает!', idEmployee;
        END IF;

        positionEmployee := (SELECT position FROM employee WHERE
        id_employee = idEmployee);

        IF (SELECT COUNT(status) FROM employee WHERE position =
        positionEmployee AND status = 'Работает') <= 1 THEN
            RAISE EXCEPTION 'Невозможно уволить единственного
            сотрудника на должности %', positionEmployee;
        END IF;

        UPDATE employee SET status = 'Не работает' WHERE id_employee
        = idEmployee;

END;
$$ LANGUAGE plpgsql
SECURITY DEFINER;

```

#### Лістинг E.16 – Створення функції gm\_employee

```

CREATE OR REPLACE FUNCTION update_application(num_application int,
numAnimals int[])
RETURNS void
AS $$
BEGIN
    -- проверка, существует ли такая заявка
    IF num_application NOT IN (SELECT keyapplication FROM
    all_applications) THEN
        RAISE EXCEPTION 'Не существует такой заявки !';
    END IF;

    -- проверка каждого животного на существование
    FOR i IN 0..array_length(numAnimals, 1) LOOP
        IF numAnimals[i] IN (SELECT номер FROM
        info_animals_writeoff) THEN
            RAISE EXCEPTION 'Животное номер % не существует или
            списано !', numAnimals[i];
        END IF;
    END LOOP;

    -- удаляем всех животных
    DELETE FROM clarification_to_app WHERE keyapplication =
    num_application;

    -- добавляем тех, которые переданы в функцию
    FOR i IN 0..array_length(numAnimals, 1) LOOP

```

```

        INSERT INTO clarification_to_app(KeyApplication,
numberanimal) VALUES (num_application, numAnimals[i]);
    END LOOP;

END;
$$ LANGUAGE plpgsql
SECURITY DEFINER;

```

### Лістинг Е.17 – Створення функції update\_application

```

CREATE OR REPLACE FUNCTION write_off_animals(numAnimals int[],
num_application int, employee_name varchar(100))
RETURNS void
AS $$
DECLARE
    employeeID int;
BEGIN

    -- проверка на существование сотрудника
    IF lower(employee_name) IN (SELECT lower(fullname) FROM
employee) THEN
        employeeID:= (SELECT id_employee FROM employee WHERE
lower(fullname) = lower(employee_name));
    ELSE
        RAISE EXCEPTION 'Не существует такого сотрудника !';
    END IF;

    -- проверка, существует ли такая заявка
    IF num_application NOT IN (SELECT keyapplication FROM
all_app_write_off) THEN
        RAISE EXCEPTION 'Не существует такой заявки на списание !';
    END IF;

    -- совпадает ли количество животных в заявке с количеством
животных поступающих в функцию
    IF array_length(numAnimals, 1) != (SELECT COUNT(numberanimal)
FROM all_app_write_off WHERE keyapplication = num_application) THEN
        RAISE EXCEPTION 'Не совпадает количество животных в заявке
!';
    END IF;

    --проверка каждого животного на нахождение в заявке
    FOR i IN 0..array_length(numAnimals, 1) LOOP
        IF numAnimals[i] NOT IN (SELECT numberanimal FROM
all_app_write_off WHERE keyapplication=num_application) THEN
            RAISE EXCEPTION 'Животное номер % не указано в заявке !',
i;
        END IF;
    END LOOP;

    -- обновляем состояния животных
    FOR i IN 1..array_length(numAnimals, 1) LOOP

```



```
        UPDATE animal SET write_off = CURRENT_DATE WHERE
numberanimal = numAnimals[i];
        INSERT INTO Inspection_history(NumberAnimal, ID_employee,
Date_inspection, Status) VALUES (numAnimals[i], employeeID,
CURRENT_DATE, 'Списанное');
        END LOOP;

        --удаляем заявку
        PERFORM rm_application(num_application);

END;
$$ LANGUAGE plpgsql
SECURITY DEFINER;
```

Лістинг Е.18 – Створення функції write\_off\_animals

## ДОДАТОК Ж. Запити на створення представлень

```
CREATE VIEW info_animals_not_writeOff AS
    SELECT DISTINCT animal.numberanimal AS номер, name_of_type AS
тип_животного, date_inspection, title AS тип_корма, status, age,
gender
    FROM animal
        JOIN type_animal ON animal.typeanimal_key =
type_animal.typeanimal_key
        JOIN feed ON animal.key_feed = feed.key_feed
        JOIN inspection_history ON animal.numberanimal =
inspection_history.numberanimal
    WHERE status != 'Списанное' and date_inspection IN (select
max(date_inspection) from inspection_history where
inspection_history.numberanimal=animal.numberanimal)
    ORDER BY animal.numberanimal;
```

### Лістинг Ж.1 – Створення представлення info\_animals\_not\_writeOff

```
CREATE VIEW info_animals_writeOff AS
    SELECT DISTINCT animal.numberanimal AS номер, name_of_type AS
тип_животного, date_inspection, title AS тип_корма, status, age,
gender
    FROM animal
        JOIN type_animal ON animal.typeanimal_key =
type_animal.typeanimal_key
        JOIN feed ON animal.key_feed = feed.key_feed
        JOIN inspection_history ON animal.numberanimal =
inspection_history.numberanimal
    WHERE status = 'Списанное' and date_inspection IN (select
max(date_inspection) from inspection_history where
inspection_history.numberanimal=animal.numberanimal)
    ORDER BY animal.numberanimal;
```

### Лістинг Ж.2 – Створення представлення info\_animals\_writeOff

```
CREATE OR REPLACE VIEW info_animals_hungry AS
    SELECT DISTINCT animal.numberanimal AS номер, name_of_type AS
тип_животного, title AS тип_корма, age, gender
    FROM animal
        JOIN type_animal ON animal.typeanimal_key =
type_animal.typeanimal_key
        JOIN feed ON animal.key_feed = feed.key_feed
        JOIN satiety_animal ON animal.numberanimal =
satiety_animal.numberanimal
    WHERE feeding_time < CURRENT_DATE - interval'1 day' AND
animal.write_off IS NULL
    ORDER BY animal.numberanimal;
```

### Лістинг Ж.3 – Створення представлення info\_animals\_hungry

```
CREATE OR REPLACE VIEW info_animals_not_hungry AS
  SELECT DISTINCT animal.numberanimal AS номер, name_of_type AS
тип_животного, title AS тип_корма, age, gender
  FROM animal
    JOIN type_animal ON animal.typeanimal_key =
type_animal.typeanimal_key
    JOIN feed ON animal.key_feed = feed.key_feed
    JOIN satiety_animal ON animal.numberanimal =
satiety_animal.numberanimal
  WHERE feeding_time > CURRENT_DATE - interval'1 day' AND
animal.write_off IS NULL
  ORDER BY animal.numberanimal;
```

### Лістинг Ж.4 – Створення представлення info\_animals\_not\_hungry

```
CREATE VIEW info_animals_sick AS
  SELECT DISTINCT animal.numberanimal AS номер, name_of_type AS
тип_животного, date_inspection, title AS тип_корма, status, age, 0
  FROM animal
    JOIN type_animal ON animal.typeanimal_key =
type_animal.typeanimal_key
    JOIN feed ON animal.key_feed = feed.key_feed
    JOIN inspection_history ON animal.numberanimal =
inspection_history.numberanimal
  WHERE status = 'Болезное' and date_inspection IN (select
max(date_inspection) from inspection_history where
inspection_history.numberanimal=animal.numberanimal)
  ORDER BY animal.numberanimal;
```

### Лістинг Ж.5 – Створення представлення info\_animals\_sick

```
CREATE VIEW info_animals_healthy AS
  SELECT DISTINCT animal.numberanimal AS номер, name_of_type AS
тип_животного, date_inspection, title AS тип_корма, status, age,
gender
  FROM animal
    JOIN type_animal ON animal.typeanimal_key =
type_animal.typeanimal_key
    JOIN feed ON animal.key_feed = feed.key_feed
    JOIN inspection_history ON animal.numberanimal =
inspection_history.numberanimal
  WHERE status = 'Здоровое' and date_inspection IN (select
max(date_inspection) from inspection_history where
inspection_history.numberanimal=animal.numberanimal)
  ORDER BY animal.numberanimal;
```

### Лістинг Ж.6 – Створення представлення info\_animals\_healthy

```
CREATE VIEW info_animals AS
```

```

SELECT numberanimal, name_of_type, title, receiptdate, age, gender,
write_off
from animal
JOIN      type_animal      ON      animal.typeanimal_key      =
type_animal.typeanimal_key
JOIN feed ON animal.key_feed = feed.key_feed
order by numberanimal;

```

### Лістинг Ж.7 – Створення представлення info\_animals

```

CREATE OR REPLACE VIEW all_app_inspection AS
SELECT
clarification_to_app.numberanimal,      application.keyapplication,
inspection_history      (SELECT      status      FROM
WHERE      numberanimal      =
clarification_to_app.numberanimal AND date_inspection IN
(SELECT      MAX(date_inspection)      FROM
inspection_history
WHERE      numberanimal      =
clarification_to_app.numberanimal GROUP BY numberanimal) ) AS status
FROM application
JOIN application_type ON application.application_type_key =
application_type.application_type_key
JOIN clarification_to_app ON application.keyapplication =
clarification_to_app.keyapplication
WHERE application_type.name_of_app::text = 'Заявка на осмотр'::text
ORDER BY application_type.name_of_app;

```

### Лістинг Ж.8 – Створення представлення all\_app\_inspection

```

CREATE OR REPLACE VIEW all_app_change_feed AS
SELECT application.keyapplication, numberanimal, name_of_app
FROM application
JOIN application_type ON application.application_type_key =
application_type.application_type_key
JOIN clarification_to_app ON application.keyapplication =
clarification_to_app.keyapplication
WHERE name_of_app = 'Заявка на изменение типа корма'

```

### Лістинг Ж.9 – Створення представлення all\_app\_change\_feed

```

CREATE OR REPLACE VIEW all_app_write_off AS
SELECT application.keyapplication, numberanimal, name_of_app
FROM application
JOIN application_type ON application.application_type_key =
application_type.application_type_key

```

```
JOIN clarification_to_app ON application.keyapplication =  
clarification_to_app.keyapplication  
WHERE name_of_app = 'Заявка на списание'
```

Лістинг Ж.10 – Створення представлення all\_app\_write\_off