**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ИС**

Курсовая РАБОТА

**по дисциплине «Управление данными»**

Тема: База данных для сотрудников кинотеатра

|  |  |  |
| --- | --- | --- |
| Студент гр. 0374 |  | Береза К.В. |
| Преподаватель |  | Татарникова Т.М. |

Санкт-Петербург

2022

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Береза К.В. | | |
| Группа 0374 | | |
| Тема работы: спроектировать базу данных, построить программу, обеспечивающую взаимодействие с ней в режиме диалога, для работников справочной службы и администраторов кинотеатров города. | | |
| Исходные данные:  Отсутствуют. | | |
| Содержание пояснительной записки:  «Содержание», «Введение», «Теоретическая часть», «Основная часть», «Заключение», «Список использованных источников», «Приложение». | | |
| Предполагаемый объем пояснительной записки:  Не менее 34 страницы. | | |
| Дата выдачи задания: 01.09.2022 | | |
| Дата сдачи реферата: 14.12.2022 | | |
| Дата защиты реферата: 14.12.2022 | | |
| Студент |  | Береза К.В. |
| Преподаватель |  | Татарникова Т.М. |

**Аннотация**

Основное содержание курсового проекта заключается в проектировании базы данных, построении программы, обеспечивающую взаимодействие с ней в режиме диалога, для работников справочной службы и администраторов кинотеатров города.

Анализ предметной области был проведён методом наблюдения. С помощью метода индукции была разработана концептуальная модель базы данных, по методу моделирования разработана программа.

При выполнении курсовой работы были следующие результаты: спроектирована база данных кинотеатров, построено программное обеспечение для работы с ней.

**Summary**

The main content of the course project is to design a database, build a program that provides interaction with it in a dialogue mode, for employees of the help desk and administrators of city cinemas.

The analysis of the subject area was carried out by the method of observation. Using the induction method, a conceptual database model was developed, and a program was developed using the modeling method.

When performing the course work, the following results were obtained: a database of cinemas was designed, software was built to work with it.

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 6](#_Toc121913344)

[**1.** **ТЕХНИЧЕСКОЕ ЗАДАНИЕ** 7](#_Toc121913345)

[1.1. Наименование программы 7](#_Toc121913346)

[1.2. Краткая характеристика области применения 7](#_Toc121913347)

[1.3. Формулировка технического задания 7](#_Toc121913348)

[**2.** **ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ** 8](#_Toc121913349)

[2.1. Основные сведения 8](#_Toc121913350)

[2.2. Концептуальная модель базы данных 8](#_Toc121913351)

[2.3. Логическая модель базы данных 9](#_Toc121913352)

[2.4. Физическая модель базы данных 10](#_Toc121913353)

[**3.** **ЗАЩИТА БАЗЫ ДАННЫХ** 15](#_Toc121913354)

[3.1. Сбой приложения 15](#_Toc121913355)

[3.2. Сбой пользовательского процесса 15](#_Toc121913356)

[3.3. Сбой процесса сервера 15](#_Toc121913357)

[3.4. Резервное копирование 15](#_Toc121913358)

[3.5. Парольная идентификация 16](#_Toc121913359)

[3.6. Определение ролей 16](#_Toc121913360)

[**4.** **ФУНКЦИОНАЛ БАЗЫ ДАННЫХ** 16](#_Toc121913361)

[4.1. Сведения о текущем состоянии проката фильмов в городе 16](#_Toc121913362)

[4.2. Манипуляция данными 17](#_Toc121913363)

[4.3. Выдача справок и отчётов 18](#_Toc121913364)

[**5.** **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ** 18](#_Toc121913365)

[5.1. Код реализованных форм на языке Python 18](#_Toc121913366)

[5.2. Графические формы 24](#_Toc121913367)

[5.3. Требования к организации входных и выходных данных 27](#_Toc121913368)

[5.4. Требования к надежности 27](#_Toc121913369)

[5.5. Условия эксплуатации 28](#_Toc121913370)

[5.6. Системные требования 29](#_Toc121913371)

[5.7. Требование к маркировке и упаковке 30](#_Toc121913372)

[5.8. Требования к транспортированию и хранению 30](#_Toc121913373)

[5.9. Специальные требования 30](#_Toc121913374)

[5.10. Стадии и этапы разработки 30](#_Toc121913375)

[5.11. Порядок контроля и приемки 31](#_Toc121913376)

[**ЗАКЛЮЧЕНИЕ** 32](#_Toc121913377)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 33](#_Toc121913378)

[**ПРИЛОЖЕНИЕ А** 34](#_Toc121913379)

# **ВВЕДЕНИЕ**

Цель работы состоит в проектировании базы данных кинотеатров и разработке программы для работы с данными в режиме диалога для сотрудников и администраторов кинотеатра. Актуальность заключается в простоте и удобстве сопровождения базы данных кинотеатров.

Задачи работы:

1. Проанализировать предметную область.
2. Спроектировать концептуальную модель базы данных.
3. Спроектировать логическую модель базы данных.
4. Спроектировать физическую модель базы данных.
5. Обеспечить целостность и защиту данных.
6. Создать программу для работы с базой данных в режиме диалога для работников справочной службы и администраторов.

# **ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

## Наименование программы

Название программы «Кинотеатры»

## Краткая характеристика области применения

Программа «Кинотеатры» предназначена для работы сотрудников справочной службы и администраторов кинотеатров города. С помощью неё можно выполнять запросы и манипулировать данными.

## Формулировка технического задания

БД должны храниться сведения о кинотеатрах: название, район, где расположен кинотеатр, категория (может быть изменена), вместимость (определяется категорией), о фильмах: назначение, производство, режиссер, жанр; кроме того, должна храниться информация о цене билета, количестве свободных мест на данный сеанс. На разных сеансах в одном кинотеатре могут идти разные фильмы. Кинотеатр может ввести новый фильм в репертуаре или снять старый с проката. Цена билета определяется прокатной стоимостью (названием) фильма и категорией кинотеатра.

Справочной службе могут потребоваться следующие сведения текущем состоянии проката фильмов в городе:

− репертуар кинотеатра (по названию кинотеатра);

− адрес и район кинотеатра (по названию кинотеатра);

− число мест (свободных) на данный сеанс (название кинотеатра и сеанс);

− цена балетов на данный сеанс (название кинотеатра и сеанс);

− жанр, производство и режиссер данного фильма (по названию);

− вместимость данного кинотеатра (по названию кинотеатра).

Администратор БД может вносить следующие изменения:

− открытие нового кинотеатра;

− снятие фильма с проката;

− изменение репертуара кинотеатра.

Необходимо предусмотреть возможность выдачи справки о сеансах фильма в указанном кинотеатре и отчета о прокате фильмов в районах города (названия фильмов, в каких кинотеатрах они демонстрировались, цена билета в каждом кинотеатре).

# **ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ**

## Основные сведения

База данных (БД) – это именованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области. БД также принято называть хранилищем данных информационной системы.

Система управления базами данных (СУБД) – совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями. Пользователи могут работать с БД средствами различных приложений, установленных на компьютере или смартфоне.

Приложения – специальные программы, с помощью которых пользователи работают с базой данных.

## Концептуальная модель базы данных

На основе технического задания определим основные сущности и их атрибуты:

1. Кинотеатры (название – первичный ключ, район, адрес, категория, вместимость).
2. Фильмы (название – первичный ключ, производство, режиссер, жанр).
3. Сеансы (номер сеанса – первичный ключ, дата, время, название кинотеатра, название фильма, количество свободных мест, цена билета).
4. Билеты (номер билета – первичный ключ, номер сеанса, ряд, место).

Функциональные требования к разрабатываемой БД:

1. На разных сеансах в одном кинотеатре могут идти разные фильмы.
2. Каждый сеанс уникален.
3. Несколько сеансов могут относиться к одному фильму.
4. Один фильм может показываться несколько раз.
5. Необходимо обеспечить хранение информации о сеансах.
6. Необходимо обеспечить хранение информации о цене билета.

Разработаем ER-модель:

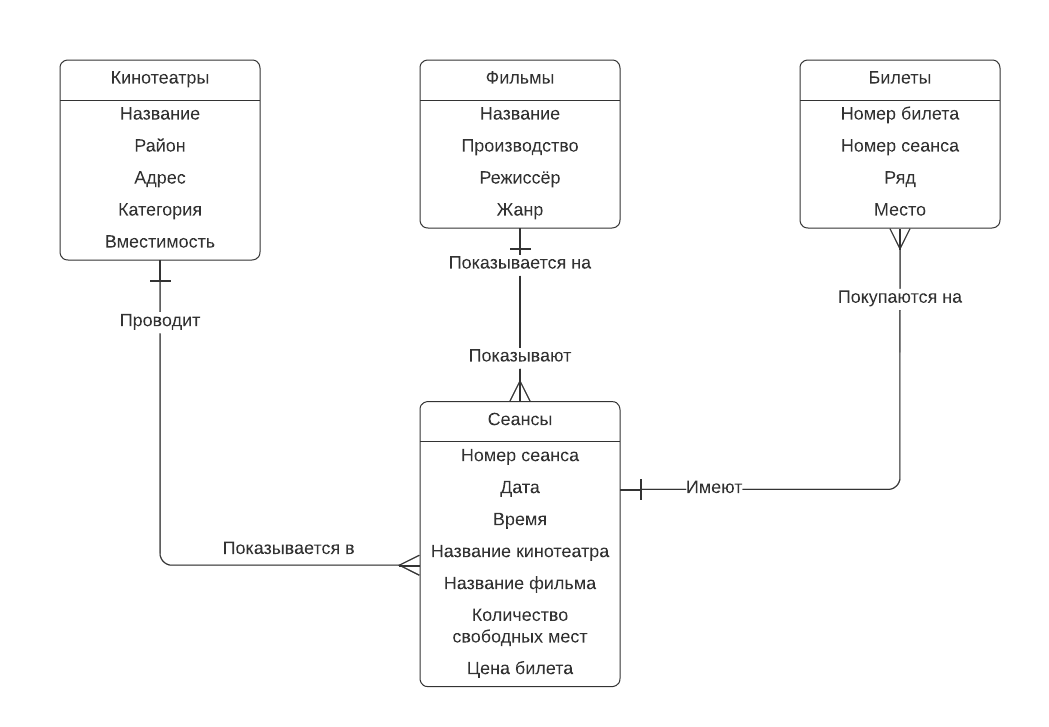


Рисунок 1 – ER-модель

## Логическая модель базы данных

В качестве модели базы данных кинотеатра была выбрана реляционная в связи с наглядностью табличного представления данных и удобства работы с ними.

На основе концептуальной модели построим логическую модель базы данных.

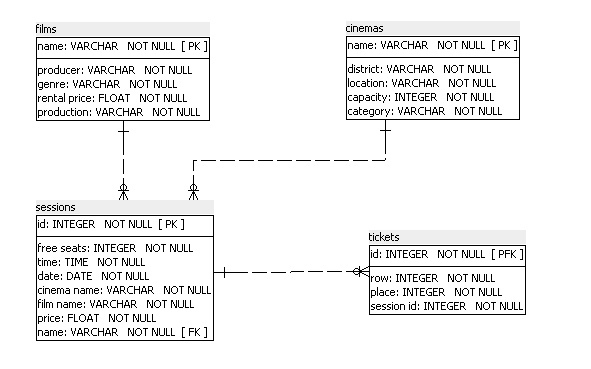


Рисунок 2 – Логическая модель базы данных

## Физическая модель базы данных

1. Для поддержания целостности данных и защиты от некорректного ввода данных определим домены для чисел (valid\_num), текстовых данных (valid\_text) и цены (valid\_cost):

create domain valid\_num as int

check (value > 0 and value is not null);

create domain valid\_cost as decimal

check (value > 0 and value is not null)

create domain valid\_text as text

check (value is not null);

1. Определим сущности:

create table cinemas(

cinema\_name text primary key,

cinema\_district valid\_text,

cinema\_location valid\_text,

cinema\_category valid\_text,

cinema\_capacity valid\_num

);

create table films(

film\_name text primary key,

film\_production valid\_text,

film\_producer valid\_text,

film\_genre valid\_text,

film\_rental\_price valid\_cost

);

Уникальность сеанса достигается путём уникального котежа, состоящего из даты, времени сеанса, названия фильма и кинотеатра, в котором идёт показ.

create table sessions(

session\_id serial primary key,

session\_time time not null,

session\_date date not null,

cinema\_name valid\_text,

film\_name valid\_text,

session\_free\_seats valid\_num,

session\_cost valid\_cost,

unique (session\_time, session\_date, cinema\_name, film\_name)

);

Уникальность билета достигается путём уникального котежа, состоящего из номера сеанса, ряда и места.

create table tickets(

ticket\_id serial primary key,

session\_id valid\_num,

ticket\_row valid\_num,

ticket\_place valid\_num,

unique (session\_id, ticket\_row, ticket\_place)

);

1. Для ускорения запросов определим индексы для каждой таблицы по её первичному ключу:

create unique index key\_cinema

on cinemas (cinema\_name);

create unique index key\_film

on films (film\_name);

create unique index key\_session

on sessions (session\_id);

create unique index key\_ticket

on tickets (ticket\_id);

1. Определим триггер на обновлении таблицы «Сеанс» поля «свободные места» по покупке билета на него:

create trigger tr\_ticket before insert on tickets

for each row execute function f\_insert\_ticket();

create or replace function f\_insert\_ticket()

returns trigger as

$$

begin

update sessions

set session\_free\_seats = session\_free\_seats - 1

where new.session\_id = session\_id;

return (new);

end;

$$ language plpgsql;

1. Определим вспомогательные функции для автозаполнения полей таблиц:

Определение вместимости кинотеатра по его категории:

create or replace function f\_capacity(cinema\_category text) returns int as $$

declare

num int;

begin

if cinema\_category = 'Кинотеатр' then num = 500;

end if;

if cinema\_category = 'Кинотеатр 3D' then num = 600;

end if;

if cinema\_category = 'Автокинотеатр' then num = 300;

end if;

return num;

end;

$$ language plpgsql;

Определение вместимости для сеанса по названию кинотеатра:

create or replace function f\_cap\_session(cinema text) returns int as $$

declare

capacity int;

begin

capacity = (select cinema\_capacity from cinemas as c

where c.cinema\_name = cinema);

return capacity;

end;

$$ language plpgsql;

Определение полной стоимости билета с учётом категории кинотеатра:

create or replace function f\_cost(cinema text, film text) returns int as $$

declare

ticket\_cost int;

category text;

begin

category = (select cinema\_category from cinemas

where cinema\_name = cinema);

if category = 'Кинотеатр' then ticket\_cost = 100;

end if;

if category = 'Кинотеатр 3D' then ticket\_cost = 150;

end if;

if category = 'Автокинотеатр' then ticket\_cost = 50;

end if;

ticket\_cost = ticket\_cost + (select film\_rental\_price from films as f

where f.film\_name = film);

return ticket\_cost;

end;

$$ language plpgsql;

# **ЗАЩИТА БАЗЫ ДАННЫХ**

## Сбой приложения

Сбой происходит при логической ошибке предложения во время его обработки (например, предложение нарушает ограничение целостности таблицы). Когда возникает сбой предложения, СУБД автоматически откатывает результаты этого предложения, генерирует сообщение об ошибке и возвращает управление пользователю (приложению пользователя).

## Сбой пользовательского процесса

Сбой пользовательского процесса – это ошибка в процессе (приложении), работающем с БД, например, аварийное разъединение или прекращение процесса. Сбившийся процесс пользователя не может продолжать работу, тогда как СУБД и процессы других пользователей могут. Система автоматически откатывает неподтверждённые транзакции сбившегося пользовательского процесса и освобождает все ресурсы, занятые этим процессом.

## Сбой процесса сервера

Сбой процесса сервера вызван проблемой, препятствующей продолжению работы сервера. Это может быть аппаратная проблема, такая как отказ питания, или программная проблема, сбой операционной системы. Восстановление после сбоя процесса сервера может потребовать перезагрузки БД, при этом автоматически происходит откат всех незавершённых транзакций.

## Резервное копирование

Резервное копирование означает периодическое сохранение файлов БД на внешнем запоминающем устройстве. Оно выполняется тогда, когда состояние файлов БД является непротиворечивым. В случае сбоя (или аварии диска) БД восстанавливается на основе последней копии.

Технология проведения резервного копирования:

− раз в неделю осуществляется полное копирование;

− раз в день частичное копирование.

## Парольная идентификация

Каждому пользователю присваиваются два параметра: имя (login) и пароль (password). При входе в систему запрашивается у пользователя его имя, а для подтверждения того, что это имя ввёл его владелец, система запрашивает пароль. Имя выдаётся пользователю при регистрации администратором, пароль пользователь устанавливает сам.

## Определение ролей

«Администратор» имеет права суперпользователя. При регистрации в базе выполняется определение роли:

set role admin

«Сотрудник» имеет только права на чтение данных. При регистрации в базе выполняется определение роли:

set role employee

# **ФУНКЦИОНАЛ БАЗЫ ДАННЫХ**

## Сведения о текущем состоянии проката фильмов в городе

Следующие запросы могут выполнять работники всех ролей («Администратор», «Сотрудник»)

Репертуар кинотеатра (по названию кинотеатра);

select cinema\_category from cinemas

where cinema\_name = 'Звезда'

Адрес и район кинотеатра (по названию кинотеатра);

select cinema\_district, cinema\_location from cinemas

where cinema\_name = 'Звезда'

Число мест (свободных) на данный сеанс (название кинотеатра и сеанс);

select session\_free\_seats from sessions

where (cinema\_name = 'Cinema park' and session\_time = '21:30'

and session\_date = '2022.11.25' and film\_name = 'Гарри Поттер')

Цена билетов на данный сеанс (название кинотеатра и сеанс);

select session\_cost from sessions

where (cinema\_name = 'Cinema park' and session\_time = '21:30'

and session\_date = '2022.11.25' and film\_name = 'Гарри Поттер')

Жанр, производство и режиссер данного фильма (по названию);

select film\_genre, film\_producer from films

where film\_name = 'Гарри Поттер'

Вместимость данного кинотеатра (по названию кинотеатра).

select cinema\_capacity from cinemas

where cinema\_name = 'Звезда'

## Манипуляция данными

Манипуляции данными могут производить работники с ролью «Администратор»

Открытие нового кинотеатра:

insert into cinemas(

values

('Меркурий', 'Санкт-Петербург', 'ул. Станюковича, д. 7', 'Кинотеатр', f\_capacity('Кинотеатр'))

);

Снятие фильма с проката:

delete from films

where film\_name = 'Колобок'

Изменение репертуара кинотеатра:

update cinemas

set cinema\_category = 'Кинотеатр 3D', cinema\_capacity = f\_capacity('Кинотеатр 3D')

where cinema\_name = 'Звезда'

## Выдача справок и отчётов

Следующие запросы могут выполнять работники всех ролей («Администратор», «Сотрудник»).

Справка о сеансах фильма в указанном кинотеатре:

select film\_name, session\_time, session\_date, session\_cost from sessions

where cinema\_name = 'Звезда'

Отчёт о прокате фильмов в районах города:

select film\_name, s.cinema\_name, session\_cost from sessions as s

inner join cinemas as c on c.cinema\_name = s.cinema\_name

where cinema\_district = 'Москва'

# **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

## Код реализованных форм на языке Python

Код стартовой формы start.py:

import sys  
from PyQt5.QtWidgets import QMainWindow, QApplication, QDateEdit, QPushButton, QLineEdit, QLabel  
from PyQt5 import uic  
import database  
  
s = dict()  
  
  
# Подкласс QMainWindow для настройки основного окна приложения  
class UIStart(QMainWindow):  
 def \_\_init\_\_(self):  
 super(UIStart, self).\_\_init\_\_()  
  
 # Загрузка ui файла  
 uic.loadUi("start.ui", self)  
 self.show()  
  
 self.login = self.findChild(QLineEdit, "lineEdit\_login")  
 self.password = self.findChild(QLineEdit, "lineEdit\_password")  
 self.password.setEchoMode(QLineEdit.Password)  
  
 self.button = self.findChild(QPushButton, "pushButton")  
 self.button.clicked.connect(self.onClicked)  
  
 def onClicked(self):  
 UIForm = database.UIForm(self.login.text(), self.password.text())  
 if UIForm.connection != -1:  
 UIForm.show()  
 s[0] = UIForm  
 self.hide()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication(sys.argv)  
 UIWindow = UIStart()  
 app.exec()

Код основной формы database.py:

import psycopg2  
from PyQt5.QtWidgets import QMessageBox, QMainWindow, QPushButton, \  
 QSpinBox, QLineEdit, QComboBox, QLabel, QTableWidget, QTableWidgetItem, QDateTimeEdit, QDialogButtonBox  
from PyQt5 import uic  
from PyQt5.QtCore import \*  
from psycopg2 import OperationalError  
import helper  
  
db = dict()  
  
  
# Подкласс QMainWindow для настройки основного окна приложения  
class UIForm(QMainWindow):  
 def \_\_init\_\_(self, login, password):  
 super(UIForm, self).\_\_init\_\_()  
  
 # Загрузка ui файла  
 uic.loadUi("main.ui", self)  
  
 # подключаемся к базе данных  
 self.connection = create\_connection(self, "postgres", login, password, "127.0.0.1", "5432")  
  
 # Инициализация объектов Категории сведений  
 self.info = self.findChild(QComboBox, "comboBox")  
 self.info.addItems(["Репертуар кинотеатра",  
 "Адрес и район кинотеатра",  
 "Число свободных мест на сеанс",  
 "Цена билетов на сеанс",  
 "Жанр, производство и режиссер фильма",  
 "Вместимость кинотеатра",  
 "Справка о сеансах фильма",  
 "Отчёт о прокате фильма"])  
 self.info.setEditable(False)  
 self.info.activated.connect(self.onActivated)  
  
 # Инициализация метки  
 self.label = self.findChild(QLabel, "label")  
 self.label2 = self.findChild(QLabel, "label\_2")  
 self.label2.hide()  
  
 self.access = self.findChild(QLabel, "label\_access")  
 self.access.setText("Уровень доступа: " + login)  
  
 # Ввод данных  
 self.input = self.findChild(QLineEdit, "lineEdit")  
  
 self.date = self.findChild(QDateTimeEdit, 'dateTimeEdit')  
 self.date.setDateTime(QDateTime.currentDateTime())  
 self.date.hide()  
  
 self.film = self.findChild(QLineEdit, "lineEdit\_film")  
 self.film.hide()  
  
 self.query = ''  
  
 # Кнопка поиск  
 self.button = self.findChild(QPushButton, 'pushButton')  
 self.button.clicked.connect(self.onClicked)  
  
 # Кнопка удаление  
 self.button\_del = self.findChild(QPushButton, 'pushButton\_delete')  
 self.button\_del.clicked.connect(self.onClicked\_del)  
  
 # Кнопка вставка  
 self.button\_ins = self.findChild(QPushButton, 'pushButton\_insert')  
 self.button\_ins.clicked.connect(self.onClicked\_ins)  
  
 # Кнопка обновления  
 self.button\_up = self.findChild(QPushButton, 'pushButton\_update')  
 self.button\_up.clicked.connect(self.onClicked\_up)  
  
 if login == 'employee':  
 self.button\_del.hide()  
 self.button\_ins.hide()  
 self.button\_up.hide()  
  
 # Вывод данных  
 self.output = self.findChild(QTableWidget, 'tableWidget')  
  
 def onClicked\_del(self):  
 db[0] = helper.UIHelper('delete', self.connection)  
 db[0].show()  
  
 def onClicked\_ins(self):  
 db[0] = helper.UIHelper('insert', self.connection)  
 db[0].show()  
  
 def onClicked\_up(self):  
 db[0] = helper.UIHelper('update', self.connection)  
 db[0].show()  
  
 def onActivated(self, ind):  
 s = 'Введите название'  
 if ind in [0, 1, 5, 6]:  
 self.label.setText(s + ' кинотеатра')  
 self.date.hide()  
 self.film.hide()  
 self.label2.hide()  
 elif ind == 4:  
 self.date.hide()  
 self.film.hide()  
 self.label2.hide()  
 self.label.setText(s + ' фильма')  
 else:  
 self.label.setText(s + ' кинотеатра')  
 self.date.show()  
 self.film.show()  
 self.label2.show()  
 if ind == 7:  
 self.label.setText(s + ' города')  
 self.date.hide()  
 self.film.hide()  
 self.label2.hide()  
  
 def onClicked(self):  
 cinema = self.input.text()  
 film = self.film.text()  
 year = self.date.date().year()  
 month = self.date.date().month()  
 day = self.date.date().day()  
  
 hour = self.date.time().hour()  
 minute = self.date.time().minute()  
  
 if self.info.currentIndex() == 0:  
 self.query = f"select cinema\_category from cinemas where cinema\_name = '{cinema}'"  
 elif self.info.currentIndex() == 1:  
 self.query = f"select cinema\_district, cinema\_location from cinemas where cinema\_name = '{cinema}'"  
 elif self.info.currentIndex() == 2:  
 self.query = f"select session\_free\_seats from sessions\  
 where(cinema\_name='{cinema}' and session\_time = '{hour}:{minute}'\  
 and session\_date = '{year}.{month}.{day}' and film\_name = '{film}')"  
 elif self.info.currentIndex() == 3:  
 self.query = f"select session\_cost from sessions\  
 where(cinema\_name='{cinema}' and session\_time = '{hour}:{minute}' \  
 and session\_date = '{year}.{month}.{day}' and film\_name = '{film}')"  
 elif self.info.currentIndex() == 4:  
 self.query = f"select film\_genre, film\_producer from films where film\_name = '{cinema}'"  
 elif self.info.currentIndex() == 5:  
 self.query = f"select cinema\_capacity from cinemas where cinema\_name = '{cinema}'"  
 elif self.info.currentIndex() == 6:  
 self.query = f"select film\_name, session\_time, session\_date, session\_cost from sessions where cinema\_name = '{cinema}'"  
 elif self.info.currentIndex() == 7:  
 self.query = f"select film\_name, s.cinema\_name, session\_cost from sessions\  
 as s inner join cinemas as c on c.cinema\_name = s.cinema\_name where cinema\_district = '{cinema}'"  
  
 self.executeTable()  
  
 def executeTable(self):  
 users = execute\_read\_query(self.connection, self.query)  
 words = self.query.split()  
 names = list()  
 for i in range(1, len(words)):  
 if words[i] == 'from':  
 break  
 else:  
 if ',' in words[i]:  
 words[i] = words[i][0:-1]  
 names.append(words[i])  
  
 self.output.setRowCount(len(users))  
 self.output.setColumnCount(len(names))  
 self.output.setHorizontalHeaderLabels(names)  
  
 for i in range(len(users)):  
 for j in range(len(names)):  
 self.output.setItem(i, j, QTableWidgetItem(str(users[i][j])))  
  
  
# подключение к базе данных  
def create\_connection(self, db\_name, db\_user, db\_password, db\_host, db\_port):  
 connection = None  
 try:  
 connection = psycopg2.connect(  
 database=db\_name,  
 user=db\_user,  
 password=db\_password,  
 host=db\_host,  
 port=db\_port,  
 )  
 print("Подключение к базе данных PostgreSQL прошло успешно")  
 except OperationalError as e:  
 print(f"Произошла ошибка '{e}'")  
 QMessageBox.information(self, 'Ошибка', "Неверный пользователь")  
 return -1  
 return connection  
  
  
# создавать таблицы, а также добавлять, изменять и удалять записи.  
def execute\_query(connection, query):  
 connection.autocommit = True  
 cursor = connection.cursor()  
 try:  
 cursor.execute(query)  
 print("Запрос выполнен успешно")  
 except OperationalError as e:  
 print(f"Произошла ошибка '{e}'")  
  
  
# выборка  
def execute\_read\_query(connection, query):  
 cursor = connection.cursor()  
 result = None  
 try:  
 cursor.execute(query)  
 result = cursor.fetchall()  
 return result  
 except OperationalError as e:  
 print(f"Произошла ошибка '{e}'")

Код вспомогательной формы helper.py:

from PyQt5.QtWidgets import QDialogButtonBox, QLineEdit, QLabel, QDialog  
from PyQt5 import uic  
import database  
  
  
# Подкласс QMainWindow для настройки основного окна приложения  
class UIHelper(QDialog):  
 def \_\_init\_\_(self, type\_=None, connection\_=None):  
 super(UIHelper, self).\_\_init\_\_()  
  
 # Загрузка ui файла  
 uic.loadUi("helper.ui", self)  
 self.connection = connection\_  
 self.type = type\_  
 self.query = ''  
  
 self.line1 = self.findChild(QLineEdit, "lineEdit\_1")  
 self.line2 = self.findChild(QLineEdit, "lineEdit\_2")  
 self.line3 = self.findChild(QLineEdit, "lineEdit\_3")  
 self.line4 = self.findChild(QLineEdit, "lineEdit\_4")  
  
 self.label1 = self.findChild(QLabel, "label\_1")  
 self.label2 = self.findChild(QLabel, "label\_2")  
 self.label3 = self.findChild(QLabel, "label\_3")  
 self.label4 = self.findChild(QLabel, "label\_4")  
  
 self.buttonBox = self.findChild(QDialogButtonBox, 'buttonBox')  
 self.buttonBox.accepted.connect(self.onClicked)  
  
 if self.type == 'insert':  
 self.label1.setText('Название кинотеатра')  
 self.label2.setText('Район')  
  
 self.line2.show()  
 self.line3.show()  
 self.line4.show()  
  
 self.label2.show()  
 self.label3.show()  
 self.label4.show()  
  
 if self.type == 'update':  
 self.label1.setText('Название кинотеатра')  
 self.label2.setText('Новая категория')  
  
 self.line2.show()  
 self.label2.show()  
  
 self.line3.hide()  
 self.line4.hide()  
 self.label3.hide()  
 self.label4.hide()  
  
 if self.type == 'delete':  
 self.label1.setText('Название фильма')  
  
 self.line2.hide()  
 self.line3.hide()  
 self.line4.hide()  
  
 self.label2.hide()  
 self.label3.hide()  
 self.label4.hide()  
  
  
 def onClicked(self):  
 if self.type == 'insert':  
 self.query = f"insert into cinemas(values('{self.line1.text()}', '{self.line2.text()}', '{self.line3.text()}', '{self.line4.text()}', f\_capacity('{self.line4.text()}')));"  
  
 if self.type == 'update':  
 self.query = f"update cinemas set cinema\_category = '{self.line2.text()}', cinema\_capacity = f\_capacity('{self.line2.text()}') where cinema\_name = '{self.line1.text()}'"  
  
 if self.type == 'delete':  
 self.query = f"delete from films where film\_name = '{self.line1.text()}'"  
  
 database.execute\_query(self.connection, self.query)  
 self.close()

## Графические формы

При запуске программы пользователь должен пройти авторизацию, по его логину определяется его роль в базе данных.

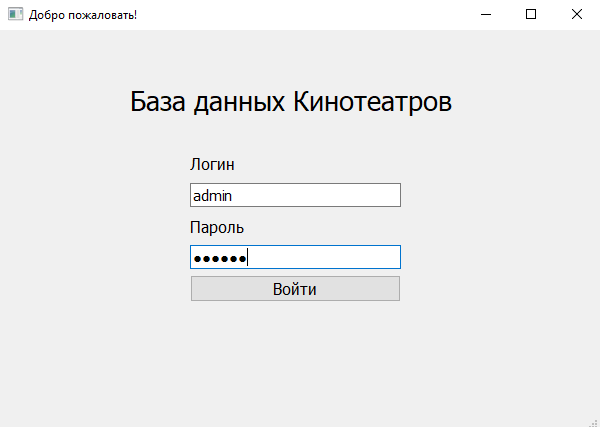


Рисунок 3 – Стартовая форма

Пример интерфейса пользователя с ролью «Администратор»:

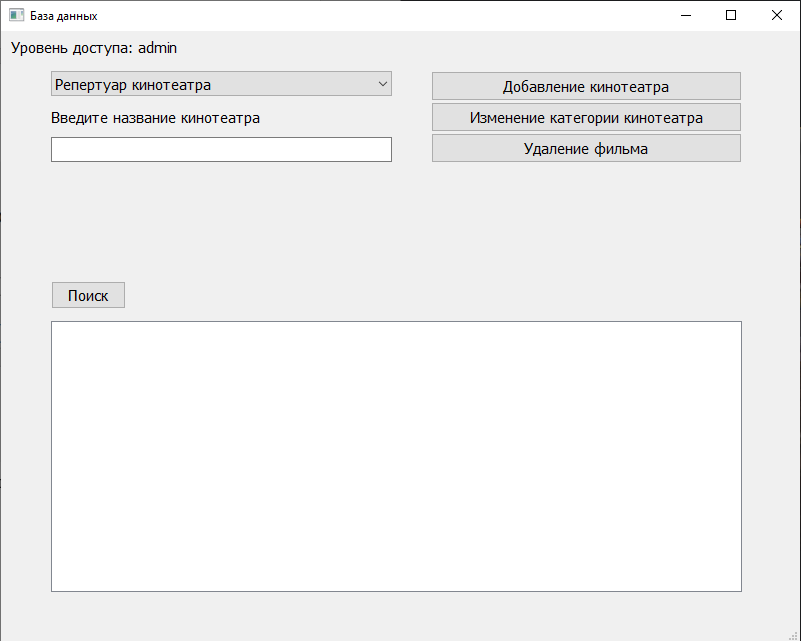


Рисунок 4 – Основная форма для роли «Администратор»

При клике на раскрывающийся список показываются запросы, которые можно выполнить с данными (идентично для роли «Сотрудник)»:

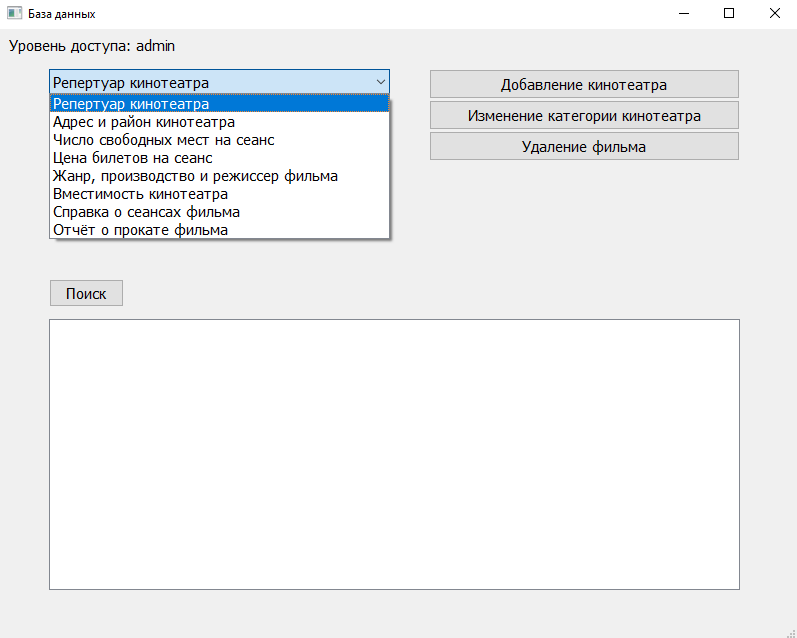


Рисунок 5 – Раскрывающийся список

Отличие функционала для роли «Администратор» состоит в том, что он может манипулировать данными, то есть добавлять, изменять или добавлять их. Это реализовано в правой части формы в виде трёх кнопок, при клике на которые всплывает дополнительное окно для ввода данных:

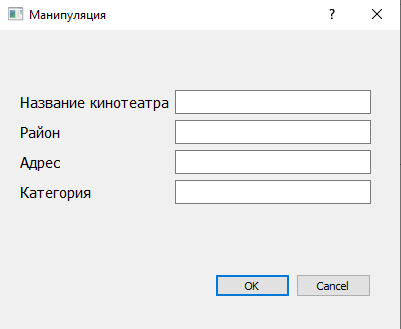


Рисунок 6 – Вспомогательная форма

Ниже представлена форма, доступная для роли «Сотрудник»:

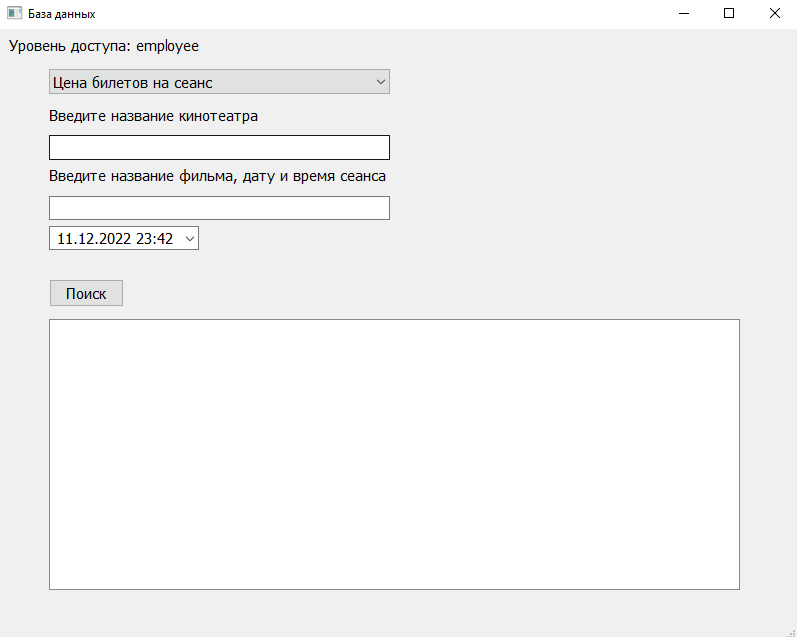


Рисунок 7 – Основная форма для роли «Сотрудник»

Пример запроса:

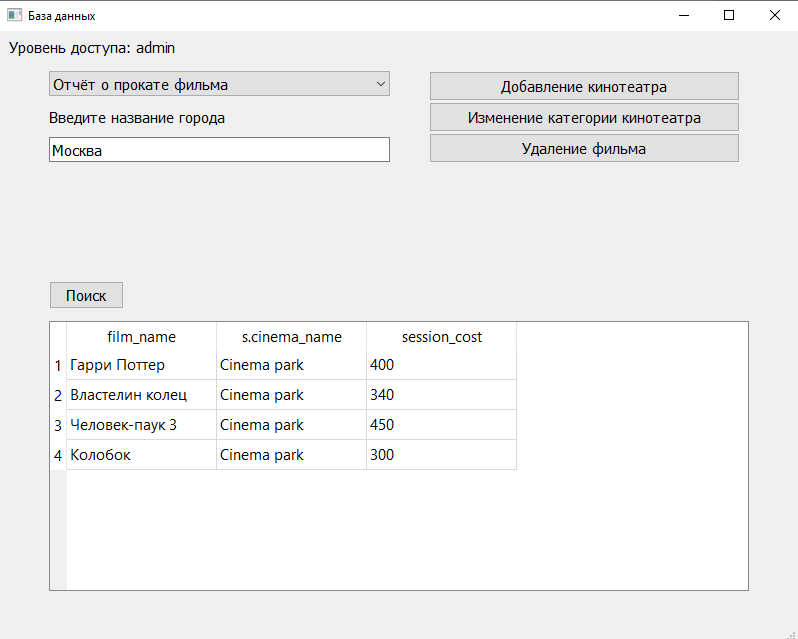


Рисунок 8 – Пример отчёта о прокате фильмов

## Требования к организации входных и выходных данных

База данных кинотеатров хранится на сервере. Поэтому необходимо подключение к Интернету, а также хорошая оперативная память.

После установки программы её можно сразу эксплуатировать.

При выборе запроса о сеансе дата и время должны быть выбраны из календаря;

Название городов, кинотеатра, фильма, района, адреса и категории — последовательность любых символов;

## Требования к надежности

Вероятность безотказной работы системы должна составлять не менее 99.99% при условии исправности сервера, базы данных и сети (связи приложения и Интернета).

#### Требования к обеспечению надежного (устойчивого) функционирования программы

Надежное (устойчивое) функционирование программы должно быть обеспечено выполнением заказчиком совокупности организационно-технических мероприятий, перечень которых приведен ниже:

* организацией бесперебойного питания технических средств;
* использованием лицензионного программного обеспечения;
* регулярным выполнением рекомендаций Министерства труда и социального развития РФ, изложенных в Постановлении от 23 июля 1998 г. «Об утверждении межотраслевых типовых норм времени на работы по сервисному обслуживанию ПЭВМ и оргтехники и сопровождению программных средств»;
* регулярным выполнением требований ГОСТ 51188-98. Защита информации. Испытания программных средств на наличие компьютерных вирусов.

#### Время восстановления после отказа

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем (не крахом) операционной системы, не должно превышать 10 минут при условии соблюдения условий эксплуатации технических и программных средств.

Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем (крахом) операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

## Условия эксплуатации

Программа (клиент) запускается на компьютере пользователя. Должна существовать устойчивая связь по сети Интернет.

Окно программы должно быть открыто, должна быть возможность закрыть, свернуть приложение или запустить любое стороннее программное обеспечение.

#### Климатические условия эксплуатации

Специальные условия не требуются.

#### Требования к видам обслуживания

Программа не требует проведения каких-либо видов обслуживания.

#### Требования к численности и квалификации персонала

При установке и настройке системы необходим IT-специалист, способный запустить установщик. В процессе эксплуатации с программой работает пользователь.

Нет особых требований к образованию IT-специалиста.

Пользователь программы должен обладать практическими навыками работы с графическим пользовательским интерфейсом операционной системы.

К квалификации пользователя специальные требования не предъявляются.

## Системные требования

Состав технических средств:

Компьютер пользователя, включающий в себя:

* процессор x64 с тактовой частотой, не менее 1 ГГц;
* оперативную память объемом, не менее 4 Гб;
* сетевое подключение выше 1 Мбит/с;
* ОС: Windows не ниже 7;
* видеокарту, монитор, мышь, клавиатура.

## Требование к маркировке и упаковке

Программное изделие передается по сети Internet в EXE файла, а также передача через твердотельные накопители. Специальных требований к маркировке не предъявляется.

## Требования к транспортированию и хранению

Специальных требований не предъявляется.

## Специальные требования

Специальных требований не предъявляется.

## Стадии и этапы разработки

Разработка должна быть проведена в три стадии:

* введение и поиск информации;
* технический (и рабочий) проекты;
* защита и демонстрация продукта.

На стадии «введение и поиск информации» должен быть выполнен этап подготовки разработки, согласования и утверждения настоящего технического задания.

На стадии «Технический (и рабочий) проект» должны быть выполнены перечисленные ниже этапы работ:

* разработка программы;
* испытания программы.

На стадии «Защита и демонстрация продукта» должны быть подготовлены презентация и демонстрационные материалы.

Содержание работ по этапам:

На этапе «Введение и поиск информации» должны быть выполнены перечисленные ниже работы:

* постановка задачи;
* определение и уточнение требований к техническим средствам;
* определение требований к программе;
* определение стадий, этапов и сроков разработки программы;
* согласование и утверждение технического задания.

На этапе разработки программы должна быть выполнена работа по программированию (кодированию) и отладке программы.

На этапе разработки программной документации должна быть выполнена разработка программных документов в соответствии с требованиями ГОСТ 19.101-77.

На этапе испытаний программы должны быть выполнены перечисленные ниже виды работ:

* разработка, согласование и утверждение порядка и методики испытаний;
* проведение приемо-сдаточных испытаний;
* корректировка программы.

## Порядок контроля и приемки

Приемосдаточные испытания программы должны проводиться согласно разработанной исполнителем и согласованной заказчиком «Программы и методики испытаний» (описанные в приложении А)

На основании результатов испытаний заказчик принимает курсовую работу.

# **ЗАКЛЮЧЕНИЕ**

В результате курсовой работы была спроектирована база данных кинотеатров, её концептуальная, логическая и физическая модели. Обеспечены целостность и конфиденциальность данных, защита базы от сбоев, созданы уровни доступа к данным.

Разработано приложение для работы с базой для работников справочной службы и администраторов кинотеатров в режиме диалога.

Полученные результаты полностью соответствуют поставленной цели, техническому заданию. Курсовой проект готов к использованию для работы с реальной базой данных кинотеатров.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

* 1. Сайт с документацией Postgresql по созданию триггеров. [Электронный ресурс] URL: https://postgrespro.ru/docs/postgresql/9.6/sql-createtrigger (Дата обращения 12.12.2022)
  2. Сайт с документацией Postgresql по созданию ролей. [Электронный ресурс] URL: https://postgrespro.ru/docs/postgresql/9.6/database-roles (Дата обращения 12.12.2022)
  3. Сайт Qt документации. [Электронный ресурс] URL: <https://doc.qt.io/qtforpython-6/> (Дата обращения 12.12.2022)

# **ПРИЛОЖЕНИЕ А**

**Методика приёмо-сдаточных испытаний**

**1. Объект испытаний**

Программное обеспечение «База данных кинотеатров», применяемое сотрудниками кинотеатров.

**2. Цель испытаний**

Проверить программное обеспечение на соответствие техническому заданию.

**3. Требования к программе**

**3.1. Требования к функциональным характеристикам**

Программа должна выполнять следующие функции:

1. Ввод информации о фильме, кинотеатре, сеансе.
2. Вывод запросов на экран.

**3.2. Требования к надёжности**

Программное обеспечение должно стабильно работать и быть отказоустойчивым, считывая пользовательский ввод.

**3.3. Требования к составу и параметрам технических средств**

Для работы программы необходимо интернет-подключение.

**3.4. Требования к информационной и программной совместимости**

Для программного обеспечения используется язык программирования Python версии 3.10, а также дополнительные библиотеки PyQt5 версии 5.15.7, psycopg2 версии 2.9.5.

* 1. **Требования к программной документации**

Программная документация содержит:

1. Техническое задание;
2. Код программы;
3. Описание программы;
4. Программу и методики испытаний;
5. Руководство пользователя;

**5. Состав и порядок испытаний**

**5.1. Тестирование ввода в форму.**

При вводе города требуется вводить города, в которых расположены кинотеатры.

При вводе кинотеатра требуется вводить только существующие кинотеатры.

При вводе фильма требуется вводить только фильмы из проката.

При вводе даты и времени можно вводить любые данные.

**5.2. Тестирование вывода**

При нажатии на кнопку «Поиск» должен выполняться запрос и выводиться на экран по раннее введённым данным.

При нажатии на кнопку «Добавление кинотеатра», «Изменение категории кинотеатра», «Удаление фильма» должна производиться манипуляция с данными.

**6. Методы испытаний**

Функционирование программы проверяется вручную.