

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет информационных технологий

ОТЧЕТ О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

по заданию: Реализация приложения базы данных в архитектуре клиент-сервер

Обучающегося: Ерошенко Дмитрия Сергеевича

группа: 21208

курс: 3

Тема задания: Информационная система кинотеатров города

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ.....	3
2. ГРАФИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ.....	4
3. КЛИЕНТСКАЯ ЧАСТЬ ИНФОРМАЦИОННОЙ СИСТЕМЫ.....	10
4. СЕРВЕРНАЯ ЧАСТЬ ИНФОРМАЦИОННОЙ СИСТЕМЫ (СУБД).....	11
5. РЕАЛИЗАЦИЯ АКТИВНОГО СЕРВЕРА. ТРИГГЕРЫ, ПРОЦЕДУРЫ, ФУНКЦИИ.....	12
6. РЕАЛИЗАЦИЯ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА.....	15
8. СИСТЕМА КОНТРОЛЯ ДОСТУПА.....	22
7. ВЫВОД.....	23
8. ПРИЛОЖЕНИЯ.....	24

1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Городская система кинотеатров представляет собой комплекс взаимосвязанных объектов, включающий фильмотеку города и отдельные кинотеатры. В фильмотеке города содержится информация о фильмах, доступных для проката в кинотеатрах. Управлением фильмотекой занимается менеджер фильмотеки. Каждый кинотеатр является информационно самодостаточным и функционирует независимо от других кинотеатров. Менеджер зданий кинотеатров отвечает за добавление новых кинотеатров в систему.

У каждого кинотеатра есть директор, который руководит персоналом, включая менеджеров и кассиров. Директор осуществляет найм и увольнение сотрудников, а также имеет доступ к списку всех работников кинотеатра. Менеджер кинотеатра отвечает за составление расписания сеансов, используя информацию о фильмах из фильмотеки города. Кассир занимается процессом продажи и отмены билетов, а также бронированием билетов по номеру телефона.

Система кинотеатров предоставляет возможность дистанционного взаимодействия с клиентами через специальную страницу, где пользователи могут просматривать сеансы в различных кинотеатрах города, производить поиск по различным фильтрам, а также получать информацию о билетах на выбранный сеанс. Пользователи могут бронировать или отменять бронь билетов по номеру телефона.

АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

В системе кинотеатров ключевыми объектами являются фильмотека города, кинотеатры, сотрудники кинотеатров (директора, менеджеры, кассиры) и пользователи (посетители). Фильмотека города выполняет центральную роль, обеспечивая кинотеатры информацией о доступных фильмах. Менеджер фильмотеки управляет этой базой данных.

Каждый кинотеатр функционирует автономно, что требует наличия менеджера зданий кинотеатров для добавления новых объектов в систему. Директор кинотеатра является ключевой фигурой в управлении персоналом, имея полномочия по найму и увольнению сотрудников. Это подразумевает наличие системы учета работников, к которой директор имеет доступ в любой момент.

Менеджер кинотеатра, составляющий расписание сеансов, зависит от информации фильмотеки, что подчеркивает важность актуальности данных в фильмотеке. Кассиры выполняют важные операционные функции, связанные с продажей билетов и взаимодействием с клиентами по вопросам бронирования и отмены билетов.

Для пользователей системы предусмотрена возможность дистанционного взаимодействия через специальную страницу, что повышает удобство использования услуг кинотеатров. Возможность поиска сеансов по различным фильтрам и управление бронированием билетов по номеру телефона делает систему гибкой и удобной для пользователей.

Таким образом, ключевыми аспектами системы являются управление информацией о фильмах, эффективное управление персоналом и обеспечение удобного доступа пользователей к услугам кинотеатров.

2. ГРАФИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Данная глава посвящена графическим представлениям создаваемой информационной системы, а именно будут рассмотрены:

1. Диаграмма прецедентов (use-case диаграмма)
2. Контекстная диаграмма
3. Диаграмма связей сущностей (ER-диаграмма)
4. Логическая схема БД
5. Принципиальная схема архитектуры приложения

2.1. Диаграмма прецедентов (use-case диаграмма)



Администратор системы может выполнять рутинные операции по созданию базы данных, а также создавать пользователей верхнего уровня - те аккаунты, которые, в теории,

достаточно зарегистрировать один раз для запуска системы в бизнесе. Также он может удалять пользователей.

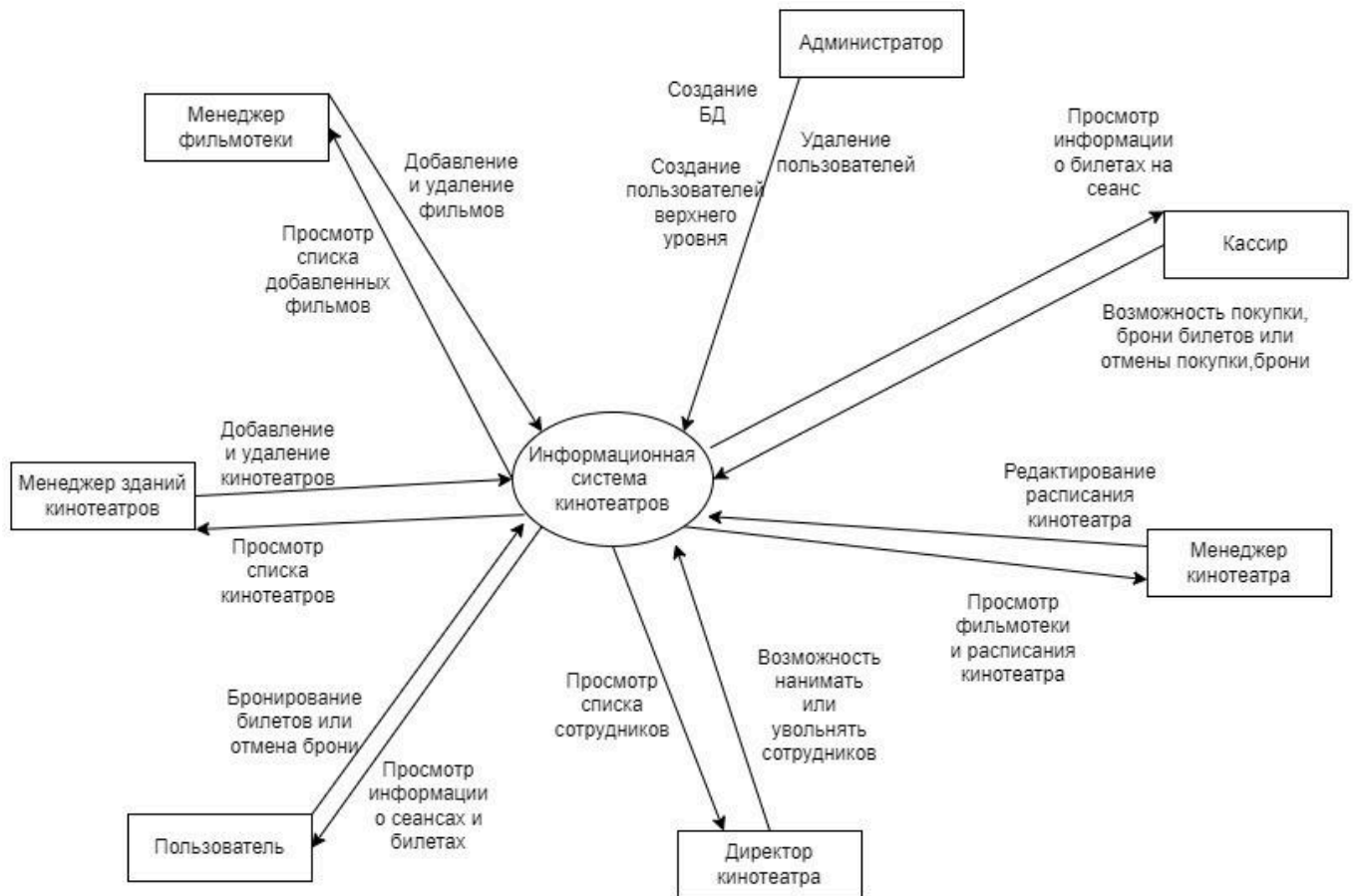
Менеджеры фильмотеки и строений кинотеатров способны изменять списки доступных фильмов и зданий кинотеатров соответственно. В кинотеатры в дальнейшем можно назначать директора, который нанимает персонал для работы, а менеджер кинотеатра может запускать фильмы из фильмотеки в прокат.

Директор, как уже было сказано, отвечает за найм и увольнение сотрудников.

Менеджер кинотеатра изменяет расписание сеансов. Он может создавать новые сеансы, выбирая фильмы из фильмотеки или удалять старые.

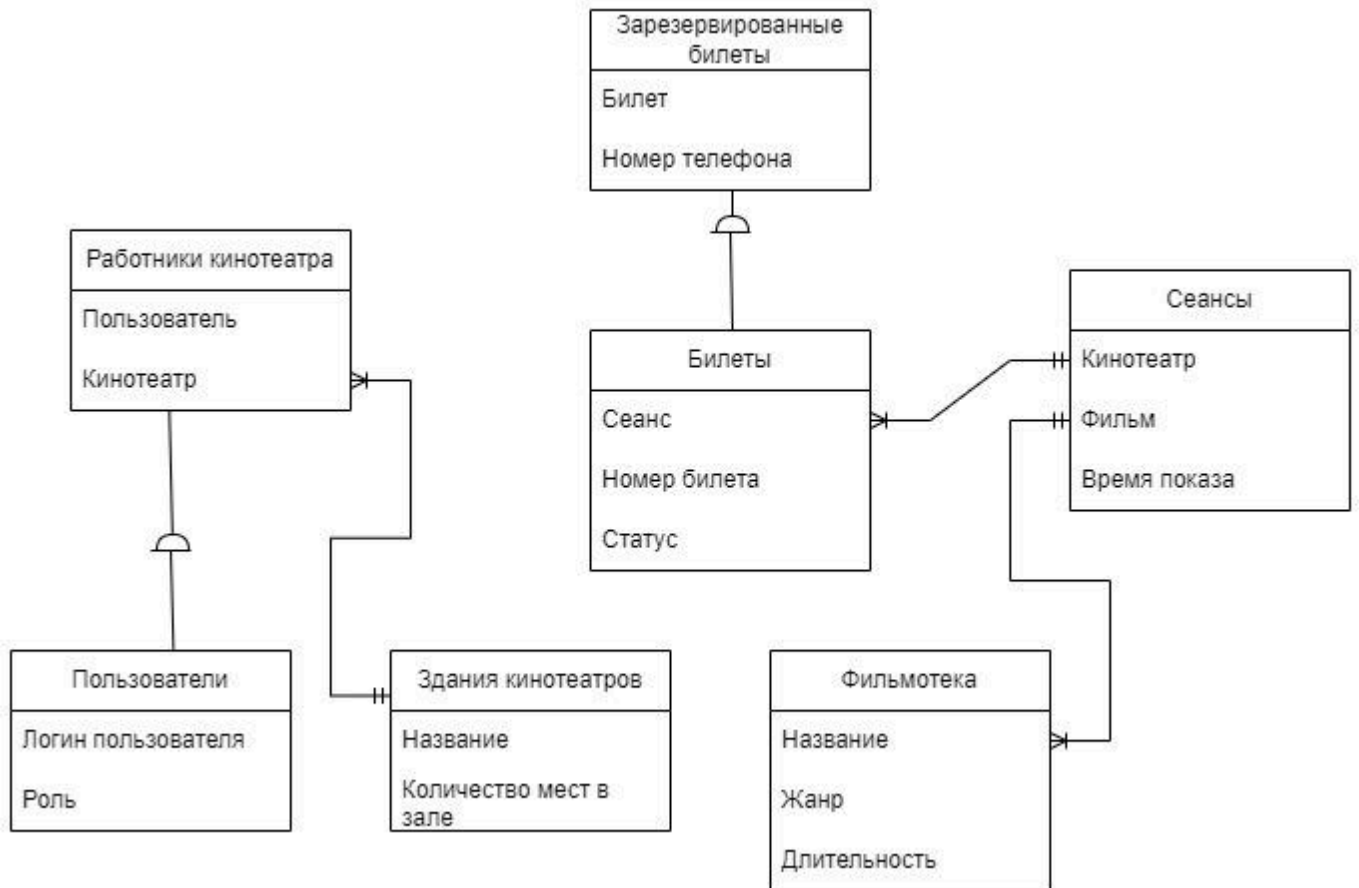
Кассир выполняет функцию торговли билетами. На кассе есть возможность купить билет, отменить покупку, забронировать билет и подтвердить бронь.

2.2. Контекстная диаграмма



Данная диаграмма показывает контекст работы каждого из пользователей. Можно увидеть явным образом потоки данных между пользователями и информационной системой, что позволило инкапсулировать большую часть этих потоков с помощью компонент активного сервера (см. раздел про активный сервер).

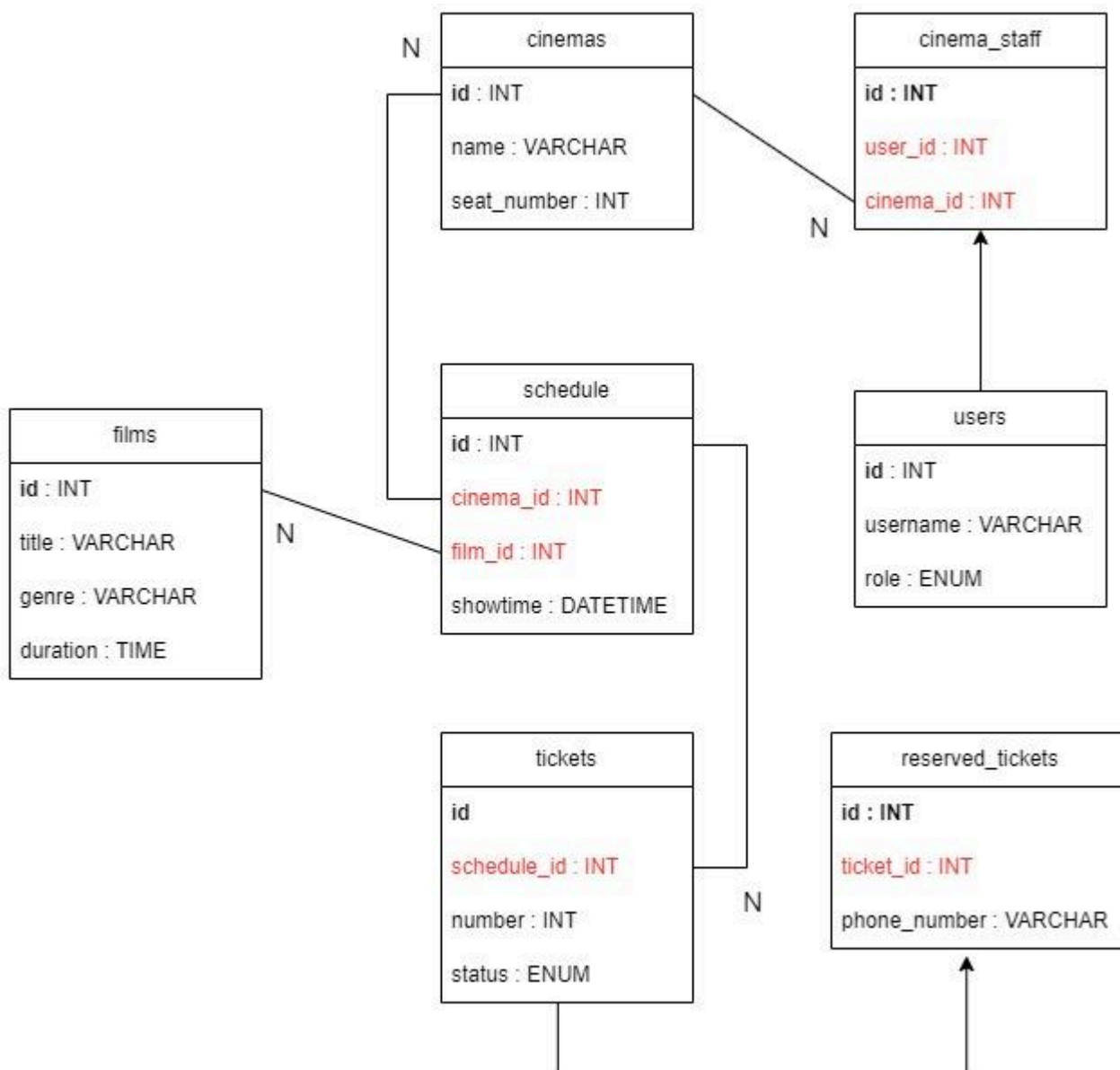
2.3. Диаграмма связи сущностей (ER-диаграмма)



В системе присутствуют пользователи, у каждого из которых есть роль. кроме того, пользователи, которые являются работниками кинотеатра, дополнительно хранят свою информацию о кинотеатре, в котором они работают. Две эти таблицы связаны при помощи узла-дискриминатора.

Есть таблица, где хранится информация о билетах, но забронированные (зарезервированные) билеты также хранят информацию о номере телефона человека, который забронировал билет. Две эти таблицы также связаны при помощи узла-дискриминатора.

2.4. Логическая схема БД



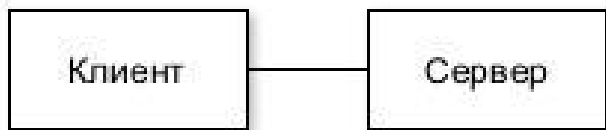
Данная схема описывает архитектуру базы данных. Можно провести однозначное соответствие с ER-диаграммой.

На диаграмме можно увидеть связи “Один ко многим”. Также есть одна связь “Многие ко многим”: это связь между фильмами и кинотеатрами. Она реализуется через 2 связи “Один ко многим” при помощи таблицы *schedule*. Важно отметить, что таблица *schedule* не является искусственно созданной для реализации такой связи. Она хранит дополнительную информацию, что делает ее неотъемлимой компонентой проектируемой Базы Данных.

Таблица *users* является супертипом для *cinema_staff*, а таблица *tickets* - супертипом для *reserved_tickets*, что отражено стрелкой и соответствует узлу-дискриминатором из ER-диаграммы.

SQL-код, описывающий концептуальную схему, приведен в **Приложениях**.

2.5. Принципиальная схема архитектуры приложения



Предполагается, что данная информационная система будет реализована в архитектуре “Клиент-сервер”, что можно увидеть на схеме выше. Клиентская часть представлена десктопным приложением с графическим пользовательским интерфейсом.

Серверная часть реализована в виде сервера базы данных. При помощи таких инструментов, как триггеры, хранимые процедуры и хранимые функции на стороне сервера была реализована большая часть бизнес-логики приложения.

3. КЛИЕНТСКАЯ ЧАСТЬ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Клиентская часть рассматриваемой информационной системы будет реализована в виде десктопного приложения на языке программирования Python. Для создания графического пользовательского интерфейса используется библиотека Tkinter, а для взаимодействия с базой данных - специальный модуль-драйвер mysql-connector-python.

4. СЕРВЕРНАЯ ЧАСТЬ ИНФОРМАЦИОННОЙ СИСТЕМЫ (СУБД)

В качестве сервера БД была выбрана система MySQL Server, а для взаимодействия с БД во время разработки для тестирования и отладки было использовано приложение MySQL Workbench. При помощи sql-скриптов была развернута база данных: созданы и инициализированы начальными значениями таблицы, объявлены триггеры, хранимые процедуры и функции. В итоговой версии программы процедуры развертывания базы данных и инициализации активного сервера автоматизированы при помощи клиентского приложения.

5. РЕАЛИЗАЦИЯ АКТИВНОГО СЕРВЕРА. ТРИГГЕРЫ, ПРОЦЕДУРЫ, ФУНКЦИИ

Триггеры.

Для связи таблиц между собой был реализован ряд триггеров.

- 1) При создании сеанса добавляются пустые билеты в таблицу tickets
- 2) При удалении сеанса удаляются билеты из таблиц tickets
- 3) При удалении билета удаляется запись из reserved_tickets если этот билет был забронирован
- 4) При удалении фильма из фильмотеки удаляются все сеансы с этим фильмом
- 5) При удалении пользователя удаляется запись из таблицы cinema_staff, если этот пользователь был работником кинотеатра
- 6) При удалении кинотеатра удаляются все пользователи, которые были работниками этого кинотеатра

Хранимые процедуры.

При помощи хранимых процедур была реализована большая часть бизнес-логики приложения. Их перечень представлен далее:

- 1) Продажа билета
- 2) Отмена продажи билета
- 3) Бронирование билета
- 4) Подтверждение брони билета
- 5) Создание пользователя - менеджера зданий кинотеатра
- 6) Создание пользователя - менеджера фильмотеки
- 7) Создание пользователя - директора кинотеатра
- 8) Просмотр списка сотрудников кинотеатра
- 9) Найм сотрудника кинотеатра
- 10) Увольнение сотрудника
- 11) Добавление сеанса в расписание кинотеатра
- 12) Удаление сеанса в расписании кинотеатра
- 13) Просмотр расписания сеансов кинотеатра
- 14) Удаление пользователя системы

Хранимые функции.

Хранимые процедуры позволяют автоматизировать получение необходимых для бизнес-логики процессов. Они активно используются как в хранимых процедурах, так и в клиентском приложении. Их перечень:

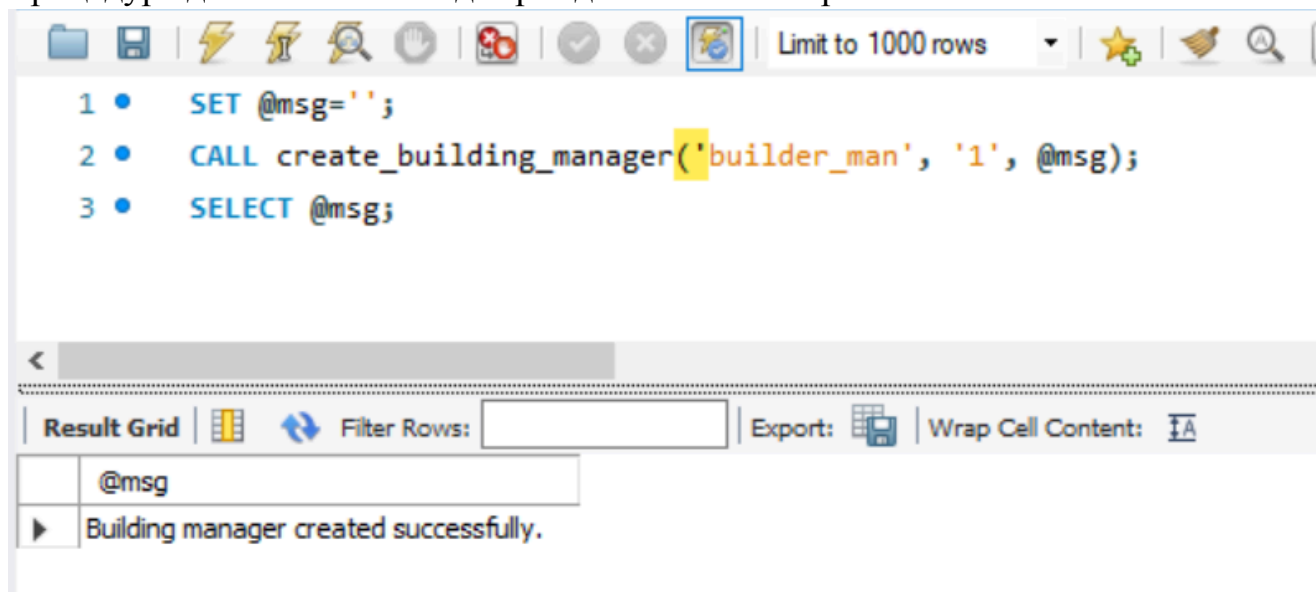
- 1) Получить количество свободных билетов на сеанс
- 2) Получить количество купленных и забронированных билетов
- 3) Узнать статус билета
- 4) Узнать id кинотеатра, которому принадлежит работник

Исходный код всех sql-скриптов для создания активного сервера можно увидеть в **Приложениях**.

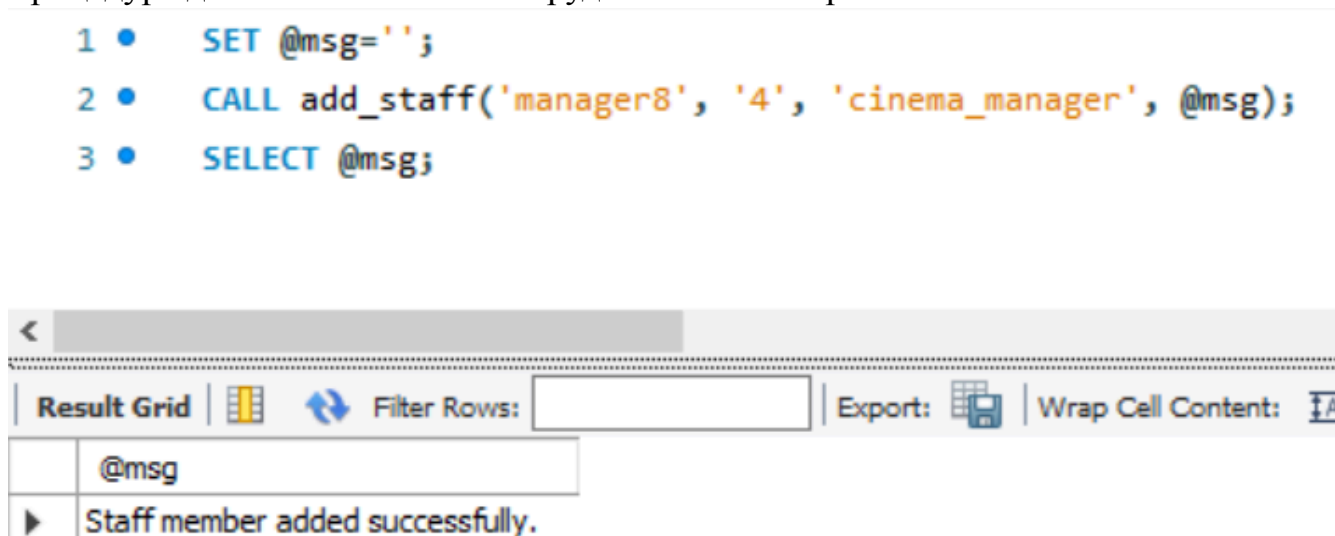
ТЕСТИРОВАНИЕ КОМПОНЕНТ ХРАНИМОГО СЕРВЕРА

Все компоненты активного сервера были проверены в приложении MySQL Workbench. Было проведено полное тестирование от инициализации базы данных и создания пользователей верхнего уровня до выполнения бизнес-процессов менеджера и кассира. Так как процедур и функций получилось умеренно много, приведено лишь несколько примеров тестирования компонент активного сервера.

Процедура добавления менеджера зданий кинотеатров.




Процедура добавления нового сотрудника кинотеатра.




Процедура продажи билета.

```
1 • SET @msg='';  
2 • CALL sell_ticket(1, 1, @msg);  
3 • SELECT @msg;
```

<	
Result Grid  Filter Rows: <input type="text"/>	
	@msg
▶	Ticket is already sold.

Процедура отмены продажи билета.

```
1 • SET @msg='';  
2 • CALL cancel_ticket(1, 1, @msg);  
3 • SELECT @msg;
```

Result Grid  Filter Rows: <input type="text"/>	
	@msg
▶	Ticket successfully canceled.

6. РЕАЛИЗАЦИЯ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА.

При запуске приложения пользователь видит страницу входа.

Вход в систему

Информационная система кинотеатров

ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ

Хост: localhost

Порт: 3306

Схема:

Логин:

Пароль:

Войти

На главной странице входа расположена инструкция пользователя, которую, при необходимости, можно прочесть.

Для начала нужно инициализировать базу данных. Для этого вводим логин и пароль пользователя СУБД **root**. **Схему вводить не нужно.**

Окно администратора

Создать схему

Использовать существующую схему

Создать менеджера культурных строений

Создать менеджера фильмотеки

Создать директора кинотеатра

Удалить пользователя

Выйти

В окне администратора можно создать БД, выбрать схему по умолчанию, создать пользователей верхнего уровня - менеджера зданий кинотеатров, менеджера фильмотеки и директора кинотеатра. При создании пользователей появляются диалоговые окна, куда нужно ввести информацию о новом персонале.

Создать менеджера культурных строений

Логин:

Пароль:

ОК

Создать директора кинотеатра

Логин:

Пароль:

Кинотеатр:

OK

При входе под ролью менеджера зданий кинотеатров появляется окно с таблицей всех кинотеатров, а также внизу расположены кнопки Добавить кинотеатр и Удалить кинотеатр.

Менеджер зданий кинотеатров

Номер кинотеатра	Название	Количество свободных мест
1	Orion	50

Добавить кинотеатр Удалить кинотеатр Выйти

Добавить кинотеатр

Название нового кинотеатра:

Количество мест в зале:

OK

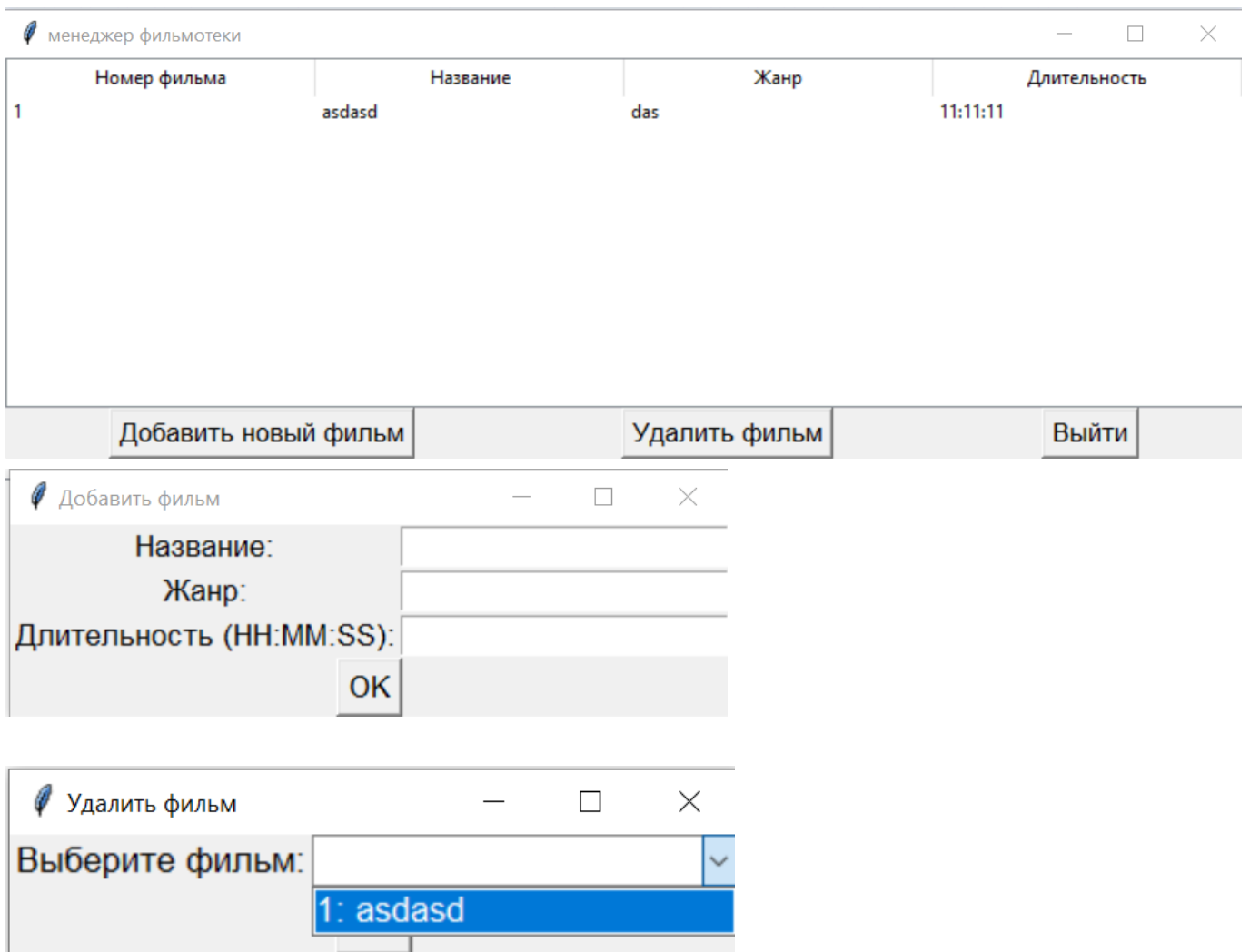
Удалить кинотеатр

Выберите кинотеатр, который хотите удалить:

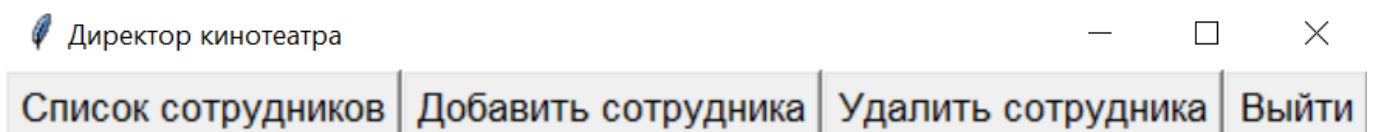
OK

1: Orion

Интерфейс менеджера фильмотеки построен аналогичным образом.



В окне директора содержатся кнопки для выполнения различных действий. При нажатии на кнопку Список сотрудников появляется окно с таблицей всех сотрудников кинотеатра. При нажатии на кнопки Добавить или Удалить сотрудника появляются окна, где нужно ввести информацию о сотрудниках.



Работники кинотеатра		—	□	×
Имя пользователя	Должность			
director	Директор			
manager	Менеджер			
cashier	Кассир			
manager8	Менеджер			

Добавить сотрудника		—	□	×
Логин:	<input type="text"/>			
Пароль:	<input type="password"/>			
Должность:	<input type="text"/> <div> Менеджер Кассир </div>			

Удаление сотрудника		—	□	×
Введите логин сотрудника:	<input type="text"/>			
	<input type="button" value="OK"/> <input type="button" value="Отмена"/>			

В окне менеджера кинотеатра есть кнопки для выполнения разных действий. При нажатии на кнопку просмотра сеансов появляется окно с таблицей с запланированными сеансами. При нажатии на кнопку просмотра фильмов появляется окно с таблицей фильмов из фильмотеки. При нажатии на кнопку добавления или удаления сеанса появляются диалоговые окна куда нужно ввести информацию.

Менеджер кинотеатра					—	□	×
Фильмотека	Добавить фильм в расписание на показ	Удалить фильм из расписания	Расписание показов	Выйти			

Фильмотека					—	□	×
Номер фильма	Название	Жанр	Длительность				
1	asdasd	das	11:11:11				

Добавить фильм в расписание

Фильм:

Дата и время показа (YYYY-MM-DD HH:MM:SS):

OK

Удалить фильм из расписания

Фильм:

Расписание

Номер показа	Название кинотеатра	Название фильма	Дата и время показа
1	Orion	asdasd	1111-11-11 11:11:11

В окне кассира есть ряд кнопок. Для начала нужно выбрать один из доступных сеансов в выпадающем списке. Затем можно просмотреть информацию о билетах на этот сеанс, создается специальное окно с графическим отображением состояния билета. Красные билеты - купленные, зеленые - свободные, желтые - забронированные. Также можно продавать билеты, отменять продажу, бронировать и подтверждать бронь. Нужно ввести номер билета и нажать соответствующую кнопку.

Кассир

Выберите показ:

Введите номер билета:

Билеты на выбранный показ

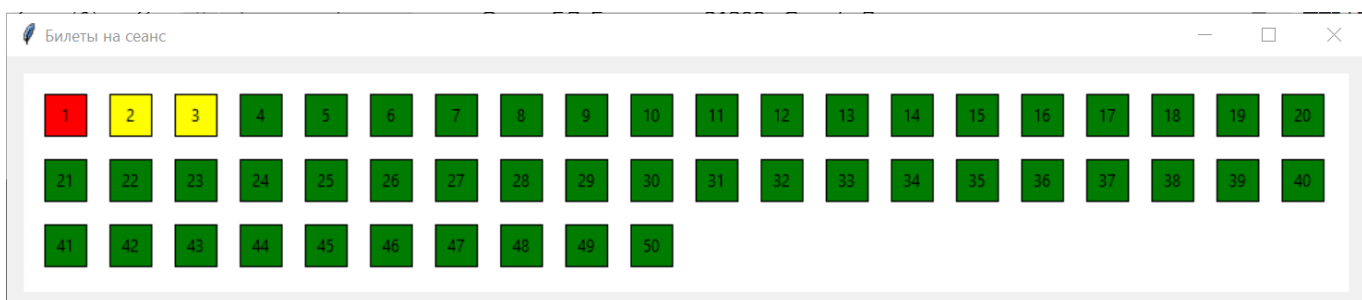
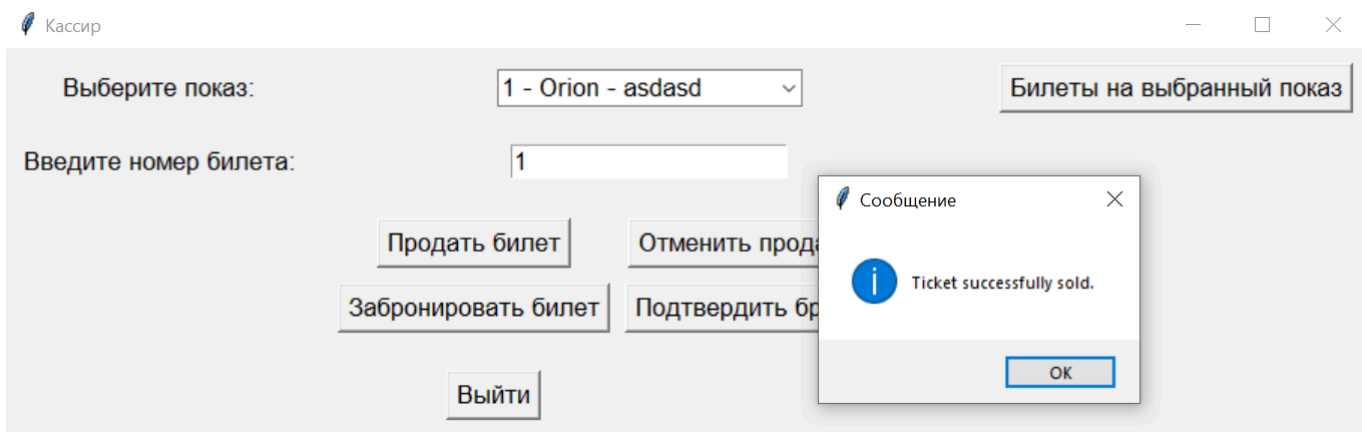
Продать билет Отменить продажу билета

Забронировать билет Подтвердить бронь билета

Выйти

Билеты на сеанс

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50										



Окно пользователя содержит ряд элементов. Вначале нужно выбрать кинотеатр, сеансы которого нас интересуют. Потом нужно выбрать один из доступных сеансов этого кинотеатра. Все это делается при помощи выпадающих списков. Потом можно посмотреть информацию о билетах на этот сеанс, а также забронировать билет или отменить бронь. Кроме того, есть кнопка для поиска сеансов. При нажатии на нее появляется окно с таблицей всех доступных сеансов в кинотеатре. Там можно производить поиск по названию фильмов, жанру и т.д. Параметр поиска выбирается при помощи радиокнопки.

Посетитель

Выберите кинотеатр: 1: Orion

Выберите сеанс: 1: asdasd

Расписание выбранного кинотеатра

Билеты на выбранный сеанс

Забронировать билет

Отменить бронь билета

Выйти

Расписание показов

Поиск:

☒ названию ☐ жанру ☐ дате и времени показа

Название	Жанр	Дата и время показа	Длительность
asdasd	das	1111-11-11 11:11:11	11:11:11

Забронировать билет

Номер билета:

Номер телефона:

OK

8. СИСТЕМА КОНТРОЛЯ ДОСТУПА

Для организации контроля доступа используются средства СУБД для работы с пользователями. Каждый пользователь приложения, кроме пользователя, представляет собой отдельного пользователя СУБД, каждый из которых обладает ограниченным набором прав.

- 1) Администратор системы авторизируется как root. Это дает ему излишний набор привилегий и чрезмерно нагружает пользователя root СУБД, что может привести к потенциальным проблемам с безопасностью. В будущем планируется реализация отдельной роли - Администратора системы, который возьмет на себя большую часть функциональных возможностей, реализованных сейчас под root.
- 2) Менеджер зданий кинотеатров обладает правами на просмотр, вставку и удаление записей в таблицу cinemas.
- 3) Менеджер фильмотеки обладает правами на просмотр, вставку и удаление записей в таблицу films.
- 4) Директор обладает правами на вызов процедур see_staff, add_staff, delete_staff.
- 5) Менеджер кинотеатра обладает правами на вызов процедур see_schedule, add_schedule, delete_schedule, просмотр таблицы films.
- 6) Кассир обладает правами на выполнение процедур sell_ticket, cancel_ticket, reserve_ticket, cancel_ticket_reservation, выполнение функций get_free_seats, get_occupied_tickets, get_ticket_status.
- 7) Пользователь обладает правами просмотра таблиц schedule, cinemas, films, процедур reserve_ticket, cancel_ticket_reservation, функций get_free_seats, get_occupied_tickets, get_ticket_status.

Кроме того, каждый пользователь обладает правами на вызов процедуры get_role, которая позволяет пользователю узнать свою роль в системе.

7. ВЫВОД

Данное приложение отвечает всем заявленным требованиям. Оно позволяет реализовывать архитектуру городской системы кинотеатров и использовать эту систему пользователям с самыми различными ролями. При помощи компонент активного сервера приложение безопасно и устойчиво к большинству атак сервера СУБД, а графический интерфейс и инструкция пользователя делают приложение user-friendly.

Данное приложение обладает некоторыми недостатками. Некоторый функционал не был реализован. Тем не менее, это приложение представляет собой законченный продукт, готовый к использованию в реальных бизнес-ситуациях.

8. ПРИЛОЖЕНИЯ

Приложение 1. SQL-код для создания концептуальной схемы

```
CREATE TABLE cinemas (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  seat_count INT NOT NULL
);

CREATE TABLE films (
  id INT AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(255) NOT NULL,
  genre VARCHAR(255) NOT NULL,
  duration TIME NOT NULL
);

CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(255) NOT NULL UNIQUE,
  role ENUM('admin', 'building_manager', 'film_manager', 'cinema_director',
'cinema_manager', 'cashier', 'user') NOT NULL
);

CREATE TABLE cinema_staff (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(255) NOT NULL,
  cinema_id INT NOT NULL,
  FOREIGN KEY (username) REFERENCES users(username),
  FOREIGN KEY (cinema_id) REFERENCES cinemas(id)
);

CREATE TABLE schedule (
  id INT AUTO_INCREMENT PRIMARY KEY,
  cinema_id INT NOT NULL,
  film_id INT NOT NULL,
  showtime DATETIME NOT NULL,
  FOREIGN KEY (cinema_id) REFERENCES cinemas(id),
  FOREIGN KEY (film_id) REFERENCES films(id)
);

CREATE TABLE tickets (
  id INT AUTO_INCREMENT PRIMARY KEY,
  schedule_id INT NOT NULL,
  seat_number INT NOT NULL,
  status ENUM('free', 'purchased', 'reserved') NOT NULL,
  FOREIGN KEY (schedule_id) REFERENCES schedule(id)
);

CREATE TABLE reserved_tickets (
  id INT AUTO_INCREMENT PRIMARY KEY,
  ticket_id INT NOT NULL,
  phone_number VARCHAR(20) NOT NULL,
  FOREIGN KEY (ticket_id) REFERENCES tickets(id)
);
```

Приложение 2. SQL-код для инициализации триггеров

```
SET SQL_SAFE_UPDATES = 0;

DELIMITER //

CREATE TRIGGER after_insert_schedule
AFTER INSERT ON schedule
FOR EACH ROW
BEGIN
    DECLARE seat_number INT DEFAULT 1;
    DECLARE total_seats INT;

    -- Получаем количество мест в кинотеатре
    SELECT seat_count INTO total_seats FROM cinemas WHERE id = NEW.cinema_id;

    -- Вставляем билеты
    WHILE seat_number <= total_seats DO
        INSERT INTO tickets (schedule_id, seat_number, status) VALUES (NEW.id,
seat_number, 'free');
        SET seat_number = seat_number + 1;
    END WHILE;
END;
//
DELIMITER ;

DELIMITER //

CREATE TRIGGER before_delete_schedule
BEFORE DELETE ON schedule
FOR EACH ROW
BEGIN
    DELETE FROM tickets WHERE schedule_id = OLD.id;
END;
//
DELIMITER ;

DELIMITER //

CREATE TRIGGER before_delete_tickets
BEFORE DELETE ON tickets
FOR EACH ROW
BEGIN
    DELETE FROM reserved_tickets WHERE ticket_id = OLD.id;
END;
//
DELIMITER ;

DELIMITER //

CREATE TRIGGER before_delete_films
BEFORE DELETE ON films
FOR EACH ROW
BEGIN
    DELETE FROM schedule WHERE film_id = OLD.id;
END;
```

```

//
DELIMITER ;

DELIMITER //

CREATE TRIGGER before_delete_users
BEFORE DELETE ON users
FOR EACH ROW
BEGIN
    DELETE FROM cinema_staff WHERE username = OLD.username;
END;
//
DELIMITER ;

DELIMITER //

CREATE TRIGGER before_delete_cinemas
BEFORE DELETE ON cinemas
FOR EACH ROW
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE user_id INT;
    DECLARE user_cursor CURSOR FOR
        SELECT user_id FROM cinema_staff WHERE cinema_id = OLD.id;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN user_cursor;

    read_loop: LOOP
        FETCH user_cursor INTO user_id;
        IF done THEN
            LEAVE read_loop;
        END IF;
        CALL delete_user((SELECT username FROM users WHERE id = user_id),
@p_message);
    END LOOP;

    CLOSE user_cursor;
END;
//
DELIMITER ;

```

Приложение 3. SQL-код для инициализации хранимых процедур

```
DELIMITER //
```

```
CREATE PROCEDURE sell_ticket(IN p_schedule_id INT, IN p_ticket_number INT,  
OUT p_message VARCHAR(255))  
BEGIN  
    DECLARE ticket_status VARCHAR(20);  
  
    SELECT status INTO ticket_status  
    FROM tickets  
    WHERE schedule_id = p_schedule_id AND seat_number = p_ticket_number;  
  
    IF ticket_status = 'free' THEN  
        UPDATE tickets  
        SET status = 'purchased'  
        WHERE schedule_id = p_schedule_id AND seat_number = p_ticket_number;  
        SET p_message = 'Ticket successfully sold.';  
    ELSEIF ticket_status = 'purchased' THEN  
        SET p_message = 'Ticket is already sold.';  
    ELSEIF ticket_status = 'reserved' THEN  
        SET p_message = 'Ticket is reserved.';  
    ELSE  
        SET p_message = 'Ticket not found.';  
    END IF;  
END;  
//  
DELIMITER ;
```

```
DELIMITER //
```

```
CREATE PROCEDURE cancel_ticket(IN p_schedule_id INT, IN p_ticket_number INT,  
OUT p_message VARCHAR(255))  
BEGIN  
    DECLARE ticket_status VARCHAR(20);  
    DECLARE ticket_id INT;  
  
    SELECT status, id INTO ticket_status, ticket_id  
    FROM tickets  
    WHERE schedule_id = p_schedule_id AND seat_number = p_ticket_number;  
  
    IF ticket_status = 'purchased' OR ticket_status = 'reserved' THEN  
        IF ticket_status = 'reserved' THEN  
            DELETE FROM reserved_tickets WHERE ticket_id = ticket_id;  
        END IF;  
  
        UPDATE tickets  
        SET status = 'free'  
        WHERE schedule_id = p_schedule_id AND seat_number = p_ticket_number;  
        SET p_message = 'Ticket successfully canceled.';  
    ELSEIF ticket_status = 'free' THEN  
        SET p_message = 'Ticket is already free.';  
    ELSE  
        SET p_message = 'Ticket not found.';  
    END IF;  
END;  
//  
DELIMITER ;
```

```

DELIMITER //
CREATE PROCEDURE confirm_reserve(IN p_schedule_id INT, IN p_ticket_number
INT, IN p_phone_number VARCHAR(20), OUT p_message VARCHAR(255))
BEGIN
    DECLARE ticket_status VARCHAR(20);
    DECLARE v_ticket_id INT;

    SELECT status, id INTO ticket_status, v_ticket_id
    FROM tickets
    WHERE schedule_id = p_schedule_id AND seat_number = p_ticket_number;

    IF ticket_status = 'reserved' THEN
        IF (SELECT phone_number FROM reserved_tickets rt
            JOIN tickets tck
            ON rt.ticket_id = tck.id
            WHERE tck.schedule_id = p_schedule_id AND
                tck.seat_number = p_ticket_number) = p_phone_number THEN
            UPDATE tickets
            SET status = 'purchased'
            WHERE schedule_id = p_schedule_id AND seat_number = p_ticket_number;

            DELETE FROM reserved_tickets
            WHERE v_ticket_id = ticket_id;

            SET p_message = 'Ticket reservation successfully confirmed.';
        ELSE
            SET p_message = 'This ticket reserved by another phone number';
        END IF;
    ELSEIF ticket_status = 'purchased' THEN
        SET p_message = 'Ticket is already sold.';
    ELSEIF ticket_status = 'free' THEN
        SET p_message = 'This ticket is free.';
    ELSE
        SET p_message = 'Ticket not found.';
    END IF;
END;
//
DELIMITER ;

DELIMITER //
CREATE PROCEDURE reserve_ticket(IN p_schedule_id INT, IN p_ticket_number INT,
IN p_phone_number VARCHAR(20), OUT p_message VARCHAR(255))
BEGIN
    DECLARE ticket_status VARCHAR(20);
    DECLARE ticket_id INT;

    SELECT status, id INTO ticket_status, ticket_id
    FROM tickets
    WHERE schedule_id = p_schedule_id AND seat_number = p_ticket_number;

    IF ticket_status = 'free' THEN
        UPDATE tickets
        SET status = 'reserved'
        WHERE schedule_id = p_schedule_id AND seat_number = p_ticket_number;

        INSERT INTO reserved_tickets (ticket_id, phone_number)

```

```

VALUES (ticket_id, p_phone_number);

SET p_message = 'Ticket successfully reserved.';
ELSEIF ticket_status = 'purchased' THEN
SET p_message = 'Ticket is already sold.';
ELSEIF ticket_status = 'reserved' THEN
SET p_message = 'Ticket is already reserved.';
ELSE
SET p_message = 'Ticket not found.';
END IF;
END;
//
DELIMITER ;

DELIMITER //
CREATE PROCEDURE create_building_manager(IN p_login VARCHAR(255), IN
p_password VARCHAR(255), OUT p_message VARCHAR(255))
BEGIN
DECLARE user_count INT;
DECLARE current_host VARCHAR(255);
DECLARE create_user_query VARCHAR(500);
DECLARE grant_privileges_query VARCHAR(500);

-- Получаем хост текущего пользователя
SET current_host = SUBSTRING_INDEX(USER(), '@', -1);

-- Проверка, существует ли пользователь с таким логином
SELECT COUNT(*) INTO user_count FROM mysql.user WHERE user = p_login AND
host = current_host;

IF user_count = 0 THEN
-- Создание пользователя в MySQL
SET @create_user_query = CONCAT('CREATE USER \'', p_login, '\''@\'',
current_host, '\'' IDENTIFIED BY \'', p_password, '\'';');
PREPARE stmt FROM @create_user_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

-- Назначение привилегий
SET @grant_privileges_query = CONCAT('GRANT SELECT, INSERT, DELETE ON
cinemas TO \'', p_login, '\''@\'', current_host, '\'';');
PREPARE stmt FROM @grant_privileges_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON FUNCTION
get_role TO \'', p_login, '\''@\'', current_host, '\'';');
PREPARE stmt FROM @grant_privileges_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

-- Вставка пользователя в таблицу users
INSERT INTO users (username, role) VALUES (p_login,
'building_manager');

SET p_message = 'Building manager created successfully.';
ELSE

```



```

        SET p_message = 'User with this login already exists.';
    END IF;
END;
//
DELIMITER ;

DELIMITER //
CREATE PROCEDURE create_film_manager(IN p_login VARCHAR(255), IN p_password
VARCHAR(255), OUT p_message VARCHAR(255))
BEGIN
    DECLARE user_count INT;
    DECLARE current_host VARCHAR(255);
    DECLARE create_user_query VARCHAR(500);
    DECLARE grant_privileges_query VARCHAR(500);

    -- Получаем хост текущего пользователя
    SET current_host = SUBSTRING_INDEX(USER(), '@', -1);

    -- Проверка, существует ли пользователь с таким логином
    SELECT COUNT(*) INTO user_count FROM mysql.user WHERE user = p_login AND
host = current_host;

    IF user_count = 0 THEN
        -- Создание пользователя в MySQL
        SET @create_user_query = CONCAT('CREATE USER \'', p_login, '\''@\'',
current_host, '\'' IDENTIFIED BY \'', p_password, '\';');
        PREPARE stmt FROM @create_user_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;

        -- Назначение привилегий
        SET @grant_privileges_query = CONCAT('GRANT SELECT, INSERT, DELETE ON
films TO \'', p_login, '\''@\'', current_host, '\';');
        PREPARE stmt FROM @grant_privileges_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;

        SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON FUNCTION
get_role TO \'', p_login, '\''@\'', current_host, '\';');
        PREPARE stmt FROM @grant_privileges_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;

        -- Вставка пользователя в таблицу users
        INSERT INTO users (username, role) VALUES (p_login, 'film_manager');

        SET p_message = 'Film manager created successfully.';
    ELSE
        SET p_message = 'User with this login already exists.';
    END IF;
END;
//
DELIMITER ;

DELIMITER //

```

```

CREATE PROCEDURE create_cinema_director(IN p_login VARCHAR(255), IN
p_password VARCHAR(255), IN p_target_cinema_id INT, OUT p_message
VARCHAR(255))
BEGIN
    DECLARE director_count INT;
    DECLARE user_count INT;
    DECLARE current_host VARCHAR(255);
    DECLARE create_user_query VARCHAR(500);
    DECLARE grant_privileges_query VARCHAR(500);

    -- Получаем хост текущего пользователя
    SET current_host = SUBSTRING_INDEX(USER(), '@', -1);

    -- Проверка, существует ли директор для данного кинотеатра
    SELECT COUNT(*) INTO director_count
    FROM cinema_staff cs
    JOIN users u ON cs.username = u.username
    WHERE cs.cinema_id = p_target_cinema_id AND u.role = 'cinema_director';

    IF director_count = 0 THEN
        -- Проверка, существует ли пользователь с таким логином
        SELECT COUNT(*) INTO user_count FROM mysql.user WHERE user = p_login
        AND host = current_host;

        IF user_count = 0 THEN
            -- Создание пользователя в MySQL
            SET @create_user_query = CONCAT('CREATE USER \'', p_login,
            '\'%\'', current_host, '\' IDENTIFIED BY \'', p_password, '\';');
            PREPARE stmt FROM @create_user_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

            -- Назначение привилегий
            SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON PROCEDURE
            see_cinema_staff TO \'', p_login, '\'%\'', current_host, '\';');
            PREPARE stmt FROM @grant_privileges_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

            SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON PROCEDURE
            add_staff TO \'', p_login, '\'%\'', current_host, '\';');
            PREPARE stmt FROM @grant_privileges_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

            SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON PROCEDURE
            delete_staff TO \'', p_login, '\'%\'', current_host, '\';');
            PREPARE stmt FROM @grant_privileges_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

            SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON FUNCTION
            get_cinema_id TO \'', p_login, '\'%\'', current_host, '\';');
            PREPARE stmt FROM @grant_privileges_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;
        END IF;
    END IF;
END

```

```

        SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON FUNCTION
get_role TO \'', p_login, '\''@\'', current_host, '\';');
        PREPARE stmt FROM @grant_privileges_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;

        -- Вставка пользователя в таблицу users
        INSERT INTO users (username, role) VALUES (p_login,
'cinema_director');
        -- Вставка записи в таблицу cinema_staff
        INSERT INTO cinema_staff (username, cinema_id)
VALUES (p_login, p_target_cinema_id);

        SET p_message = 'Cinema director created successfully.';
    ELSE
        SET p_message = 'User with this login already exists.';
    END IF;
ELSE
    SET p_message = 'This cinema already has a director.';
END IF;
END;
//
DELIMITER ;

DELIMITER //
CREATE PROCEDURE see_cinema_staff(IN p_target_cinema_id INT)
BEGIN
    SELECT u.username, u.role
    FROM cinema_staff cs
    JOIN users u ON cs.username = u.username
    WHERE cs.cinema_id = p_target_cinema_id;
END;
//
DELIMITER ;

DELIMITER //
CREATE PROCEDURE add_staff(IN p_login VARCHAR(255), IN p_password
VARCHAR(255), IN p_role VARCHAR(255), OUT p_message VARCHAR(255))
BEGIN
    DECLARE user_count INT;
    DECLARE current_host VARCHAR(255);
    DECLARE create_user_query VARCHAR(500);
    DECLARE grant_privileges_query VARCHAR(500);

    -- Получаем хост текущего пользователя
    SET current_host = SUBSTRING_INDEX(USER(), '@', -1);

    -- Проверка, существует ли пользователь с таким логином
    SELECT COUNT(*) INTO user_count FROM mysql.user WHERE user = p_login AND
host = current_host;

    IF user_count = 0 THEN
        -- Проверка роли
        IF p_role NOT IN ('cinema_manager', 'cashier') THEN
            SET p_message = CONCAT('There is no such position in the cinema:
', p_role);
        ELSE

```

```

-- Создание пользователя в MySQL
SET @create_user_query = CONCAT('CREATE USER \'', p_login,
\'\'@\'', current_host, \'\' IDENTIFIED BY \'', p_password, \'\'');
PREPARE stmt FROM @create_user_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

-- Назначение привилегий
IF p_role = 'cinema_manager' THEN
SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE see_schedule TO \'', p_login, \'\'@\'', current_host, \'\'');
PREPARE stmt FROM @grant_privileges_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE add_schedule TO \'', p_login, \'\'@\'', current_host, \'\'');
PREPARE stmt FROM @grant_privileges_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE delete_schedule TO \'', p_login, \'\'@\'', current_host, \'\'');
PREPARE stmt FROM @grant_privileges_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
FUNCTION get_cinema_id TO \'', p_login, \'\'@\'', current_host, \'\'');
PREPARE stmt FROM @grant_privileges_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
FUNCTION get_role TO \'', p_login, \'\'@\'', current_host, \'\'');
PREPARE stmt FROM @grant_privileges_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

ELSEIF p_role = 'cashier' THEN
SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE sell_ticket TO \'', p_login, \'\'@\'', current_host, \'\'');
PREPARE stmt FROM @grant_privileges_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE cancel_ticket TO \'', p_login, \'\'@\'', current_host, \'\'');
PREPARE stmt FROM @grant_privileges_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE reserve_ticket TO \'', p_login, \'\'@\'', current_host, \'\'');
PREPARE stmt FROM @grant_privileges_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

```

```

        SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE confirm_reserve TO \'', p_login, '\\'\@\\', current_host, '\\';');
        PREPARE stmt FROM @grant_privileges_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;

        SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE see_schedule TO \'', p_login, '\\'\@\\', current_host, '\\';');
        PREPARE stmt FROM @grant_privileges_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;

        SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
FUNCTION get_cinema_id TO \'', p_login, '\\'\@\\', current_host, '\\';');
        PREPARE stmt FROM @grant_privileges_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;

        SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
FUNCTION count_free_seats TO \'', p_login, '\\'\@\\', current_host, '\\';');
        PREPARE stmt FROM @grant_privileges_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;

        SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
FUNCTION count_occupied_tickets TO \'', p_login, '\\'\@\\', current_host,
'\\';');
        PREPARE stmt FROM @grant_privileges_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;

        SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
FUNCTION get_ticket_status TO \'', p_login, '\\'\@\\', current_host, '\\';');
        PREPARE stmt FROM @grant_privileges_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;

        SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
FUNCTION get_role TO \'', p_login, '\\'\@\\', current_host, '\\';');
        PREPARE stmt FROM @grant_privileges_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;
    END IF;

    -- Вставка пользователя в таблицу users
    INSERT INTO users (username, role) VALUES (p_login, p_role);
    -- Вставка записи в таблицу cinema_staff
    INSERT INTO cinema_staff (username, cinema_id)
    VALUES (p_login, (SELECT get_cinema_id()));

    SET p_message = 'Staff member added successfully.';
END IF;
ELSE
    SET p_message = 'User with this login already exists.';
END IF;
END;

```

```

//
DELIMITER ;

DELIMITER //
CREATE PROCEDURE delete_staff(IN p_login VARCHAR(255), OUT p_message
VARCHAR(255))
BEGIN
    DECLARE user_count INT;
    DECLARE cinema_id INT;
    DECLARE current_host VARCHAR(255);
    DECLARE drop_user_query VARCHAR(500);

    -- Получаем хост текущего пользователя
    SET current_host = SUBSTRING_INDEX(USER(), '@', -1);

    -- Проверка, существует ли пользователь с таким логином
    SELECT COUNT(*) INTO user_count FROM mysql.user WHERE user = p_login AND
host = current_host;

    IF user_count = 0 THEN
        SET p_message = 'User with this login does not exist.';
    ELSE
        -- Проверка, является ли пользователь сотрудником данного кинотеатра
        SELECT cs.cinema_id INTO cinema_id
        FROM cinema_staff cs
        WHERE cs.username = p_login;

        IF cinema_id != (SELECT cs.cinema_id
                        FROM cinema_staff cs
                        WHERE cs.username = SUBSTRING_INDEX(USER(), '@', 1))
THEN
            SET p_message = 'This user does not belong to your cinema.';
        ELSE
            -- Удаление пользователя из MySQL
            SET @drop_user_query = CONCAT('DROP USER \'', p_login, '\''@\'',
current_host, '\';');
            PREPARE stmt FROM @drop_user_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

            -- Удаление записи из таблиц users и cinema_staff
            DELETE FROM users WHERE username = p_login;
            DELETE FROM cinema_staff WHERE username = p_login;

            SET p_message = 'Staff member deleted successfully.';
        END IF;
    END IF;
END;
//
DELIMITER ;

DELIMITER //
CREATE PROCEDURE add_schedule(IN p_target_film_id INT, IN p_target_showtime
DATETIME, OUT p_message VARCHAR(255))
BEGIN
    DECLARE cinema_id INT;

```

```

-- Получение cinema_id для данного пользователя
SET cinema_id = get_cinema_id();

-- Вставка новой записи в расписание
INSERT INTO schedule (cinema_id, film_id, showtime)
VALUES (cinema_id, p_target_film_id, p_target_showtime);

SET p_message = 'Schedule added successfully.';
END;
//
DELIMITER ;

DELIMITER //
CREATE PROCEDURE delete_schedule(IN p_schedule_id INT, OUT p_message
VARCHAR(255))
BEGIN
    DECLARE schedule_cinema_id INT;
    DECLARE user_cinema_id INT;

    -- Получение cinema_id для данной записи расписания
    SELECT cinema_id INTO schedule_cinema_id FROM schedule WHERE id =
p_schedule_id;

    -- Получение cinema_id текущего пользователя
    SELECT cs.cinema_id INTO user_cinema_id
    FROM cinema_staff cs
    WHERE cs.username = SUBSTRING_INDEX(USER(), '@', 1);

    IF schedule_cinema_id = user_cinema_id THEN
        -- Удаление записи из расписания
        DELETE FROM schedule WHERE id = p_schedule_id;
        SET p_message = 'Schedule deleted successfully.';
    ELSE
        SET p_message = 'Schedule not found or you do not have permission to
delete it.';
    END IF;
END;
//
DELIMITER ;

DELIMITER //
CREATE PROCEDURE see_schedule()
BEGIN
    SELECT s.id, c.name AS cinema_name, f.title AS film_title, s.showtime
    FROM schedule s
    JOIN cinemas c ON s.cinema_id = c.id
    JOIN films f ON s.film_id = f.id
    WHERE s.cinema_id = (SELECT cs.cinema_id
                        FROM cinema_staff cs
                        WHERE cs.username = SUBSTRING_INDEX(USER(), '@', 1));
END;
//
DELIMITER ;

DELIMITER //
CREATE PROCEDURE see_tickets()
BEGIN

```

```

SELECT t.id, t.schedule_id, t.seat_number, t.status
FROM tickets t
JOIN schedule s ON t.schedule_id = s.id
WHERE s.cinema_id = (SELECT cs.cinema_id
                     FROM cinema_staff cs
                     WHERE cs.username = SUBSTRING_INDEX(USER(), '@', 1));
END;
//
DELIMITER ;

DELIMITER //
CREATE PROCEDURE delete_user(IN p_login VARCHAR(255), OUT p_message
VARCHAR(255))
BEGIN
    DECLARE user_id INT;
    DECLARE user_role ENUM('admin', 'building_manager', 'film_manager',
'cinema_director', 'cinema_manager', 'cashier', 'user');
    DECLARE current_host VARCHAR(255);
    DECLARE drop_user_query VARCHAR(500);

    -- Получаем хост текущего пользователя
    SET current_host = SUBSTRING_INDEX(USER(), '@', -1);

    -- Проверка, существует ли пользователь с таким логином
    SELECT id, role INTO user_id, user_role FROM users WHERE username =
p_login;

    IF user_id IS NULL THEN
        SET p_message = 'User with this login does not exist.';
    ELSE
        -- Проверка, является ли пользователь кассиром, директором или
менеджером кинотеатра
        IF user_role IN ('cinema_director', 'cinema_manager', 'cashier') THEN
            -- Удаление записи из таблицы cinema_staff
            DELETE FROM cinema_staff WHERE username = p_login;
        END IF;

        -- Удаление записи из таблицы users
        DELETE FROM users WHERE username = p_login;

        -- Удаление пользователя из MySQL
        SET @drop_user_query = CONCAT('DROP USER \'', p_login, '\'@\'',
current_host, '\';');
        PREPARE stmt FROM @drop_user_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;

        SET p_message = 'User deleted successfully.';
    END IF;
END;
//
DELIMITER ;

```


Приложение 4. SQL-код для инициализации хранимых функций

```
DELIMITER //
```

```
CREATE FUNCTION count_free_seats(p_schedule_id INT) RETURNS INT
READS SQL DATA
BEGIN
    DECLARE free_seats INT;

    SELECT COUNT(*) INTO free_seats
    FROM tickets
    WHERE schedule_id = p_schedule_id AND status = 'free';

    RETURN free_seats;
END;
//
DELIMITER ;

DELIMITER //
```

```
CREATE FUNCTION count_occupied_tickets(p_schedule_id INT) RETURNS INT
READS SQL DATA
BEGIN
    DECLARE occupied_tickets INT;

    SELECT COUNT(*) INTO occupied_tickets
    FROM tickets
    WHERE schedule_id = p_schedule_id AND status IN ('purchased', 'reserved');

    RETURN occupied_tickets;
END;
//
DELIMITER ;

DELIMITER //
```

```
CREATE FUNCTION get_ticket_status(p_schedule_id INT, p_ticket_number INT)
RETURNS VARCHAR(20)
READS SQL DATA
BEGIN
    DECLARE ticket_status VARCHAR(20);

    SELECT status INTO ticket_status
    FROM tickets
    WHERE schedule_id = p_schedule_id AND seat_number = p_ticket_number;

    IF ticket_status IS NULL THEN
        RETURN 'Ticket not found';
    ELSE
        RETURN ticket_status;
    END IF;
END;
//
DELIMITER ;

DELIMITER //
```

```
CREATE FUNCTION get_cinema_id() RETURNS INT
READS SQL DATA
BEGIN
```

```

        RETURN (SELECT cinema_id
                  FROM cinema_staff cs
                  WHERE cs.username = SUBSTRING_INDEX(USER(), '@', 1));
END;
//
DELIMITER ;

DELIMITER //
CREATE FUNCTION get_role() RETURNS VARCHAR(255)
READS SQL DATA
BEGIN
    RETURN (SELECT role
            FROM users us
            WHERE us.username = SUBSTRING_INDEX(USER(), '@', 1));
END;
//
DELIMITER ;

```

Приложение 5. Код программы клиентского приложения на языке Python.

```
import tkinter as tk
from tkinter import messagebox, simpledialog, ttk
from tkinter import font as tkfont
import mysql.connector
from mysql.connector import Error

db_params = dict()
db_params['host'] = 'localhost'
db_params['port'] = 3306

font_size = 13

class LoginWindow:
    def __init__(self, master):
        self.master = master
        self.master.title("Вход в систему")

        self.title_label = tk.Label(master, text="Информационная  
система кинотеатров")
        self.title_label.grid(row=0, column=0)

        self.host_label = tk.Label(master, text="Хост:")
        self.host_label.grid(row=1, column=0)
        self.host_entry = tk.Entry(master)
        self.host_entry.insert(0, db_params.get('host', ''))
        self.host_entry.grid(row=1, column=1)

        self.port_label = tk.Label(master, text="Порт:")
        self.port_label.grid(row=2, column=0)
        self.port_entry = tk.Entry(master)
        self.port_entry.insert(0, db_params.get('port', ''))
        self.port_entry.grid(row=2, column=1)

        self.schema_label = tk.Label(master, text="Схема:")
        self.schema_label.grid(row=3, column=0)
        self.schema_entry = tk.Entry(master)
        self.schema_entry.insert(0, db_params.get('database', ''))
        self.schema_entry.grid(row=3, column=1)

        self.login_label = tk.Label(master, text="Логин:")
        self.login_label.grid(row=4, column=0)
        self.login_entry = tk.Entry(master)
        self.login_entry.grid(row=4, column=1)

        self.password_label = tk.Label(master, text="Пароль:")
        self.password_label.grid(row=5, column=0)
        self.password_entry = tk.Entry(master, show="*")
        self.password_entry.grid(row=5, column=1)
```

```

        self.login_button = tk.Button(master, text="Войти",
command=self.login)
        self.login_button.grid(row=6, columnspan=2)

        self.instruction_button = tk.Button(master, text="ИНСТРУКЦИЯ
ПОЛЬЗОВАТЕЛЯ", font=tkfont.Font(size=20),
command=self.show_instructions)
        self.instruction_button.grid(row=0, column=1)

    def login(self):
        host = self.host_entry.get()
        port = self.port_entry.get()
        login = self.login_entry.get()
        password = self.password_entry.get()

        if not host or not port or not login or not password:
            messagebox.showinfo("Ошибка", "Все поля должны быть
заполнены")
            return

        db_params['host'] = host
        db_params['port'] = port
        db_params['user'] = login
        db_params['password'] = password
        schema = self.schema_entry.get()
        if schema != '':
            db_params['database'] = schema

        try:
            connection = mysql.connector.connect(
                host=host,
                port=port,
                user=login,
                password=password
            )
            if connection.is_connected():
                cursor = connection.cursor()
                if login == "root":
                    self.master.destroy()
                    root_window = tk.Tk()
                    default_font = tkfont.Font(family="Helvetica",
size=font_size, weight="normal")
                    root_window.option_add("*Font", default_font)
                    RootWindow(root_window, connection) # Pass
connection

                    root_window.mainloop()
                else:
                    cursor.execute(f"USE {db_params['database']}")
                    cursor.execute("SELECT get_role();")
                    role = cursor.fetchone()[0]
                    if role:
                        self.master.destroy()

```

```

        if role == "building_manager":
            bm_window = tk.Tk()

            default_font =
tkfont.Font(family="Helvetica", size=font_size, weight="normal")
            bm_window.option_add("*Font", default_font)
            BuildingManagerWindow(bm_window,
connection) # Pass connection
            bm_window.mainloop()
        elif role == "film_manager":
            fm_window = tk.Tk()

            default_font =
tkfont.Font(family="Helvetica", size=font_size, weight="normal")
            fm_window.option_add("*Font", default_font)
            FilmManagerWindow(fm_window, connection) #
Pass connection
            fm_window.mainloop()
        elif role == "cinema_director":
            cd_window = tk.Tk()

            default_font =
tkfont.Font(family="Helvetica", size=font_size, weight="normal")
            cd_window.option_add("*Font", default_font)
            CinemaDirectorWindow(cd_window, connection)
# Pass connection
            cd_window.mainloop()
        elif role == "cinema_manager":
            cm_window = tk.Tk()

            default_font =
tkfont.Font(family="Helvetica", size=font_size, weight="normal")
            cm_window.option_add("*Font", default_font)
            CinemaManagerWindow(cm_window, connection)
# Pass connection
            cm_window.mainloop()
        elif role == "cashier":
            cashier_window = tk.Tk()

            default_font =
tkfont.Font(family="Helvetica", size=font_size, weight="normal")
            cashier_window.option_add("*Font",
default_font)
            CashierWindow(cashier_window, connection)
# Pass connection
            cashier_window.mainloop()
        elif role == "user":
            user_window = tk.Tk()

            default_font =
tkfont.Font(family="Helvetica", size=font_size, weight="normal")
            user_window.option_add("*Font",
default_font)
            UserWindow(user_window, connection) # Pass
connection
            user_window.mainloop()
        else:
            messagebox.showinfo("Ошибка", "У данного
пользователя неизвестная роль")

```

```

        else:
            messagebox.showinfo("Ошибка", "пользователь не найден")

        cursor.close()
    except Error as e:
        messagebox.showinfo("Ошибка:", f"MySql: {e}")

    def show_instructions(self):
        instructions_window = tk.Toplevel(self.master)
        instructions_window.title("Инструкция пользователя")
        InstructionWindow(instructions_window)

class InstructionWindow:
    def __init__(self, master):
        self.master = master
        self.master.title("Инструкция пользователя")

        self.canvas = tk.Canvas(master)
        self.scrollbar = tk.Scrollbar(master, orient="vertical",
command=self.canvas.yview)
        self.scrollable_frame = tk.Frame(self.canvas)

        self.scrollable_frame.bind(
            "<Configure>",
            lambda e: self.canvas.configure(
                scrollregion=self.canvas.bbox("all")
            )
        )

        self.canvas.create_window((0, 0), window=self.scrollable_frame,
anchor="nw")
        self.canvas.configure(yscrollcommand=self.scrollbar.set)

        self.canvas.pack(side="left", fill="both", expand=True)
        self.scrollbar.pack(side="right", fill="y")

        self.intro_text = tk.Label(self.scrollable_frame, text=(
            "Добро пожаловать в информационную систему кинотеатров!\n"
            "Чтобы войти в систему, вам необходимо указать адрес вашего сервера (поле 'Хост'),\n"
            "порт, к которому будет произведено подключение (поле 'Порт'), имя вашей базы данных (поле 'Схема'),\n"
            "а также ваши логин и пароль в системе (Поля 'Логин' и 'Пароль') .\n"
            "После успешного входа вы сможете работать!\n"
            "Для начала давайте определимся с вашей ролью в системе."
        ))
        self.intro_text.pack(pady=10)

        self.admin_button = tk.Button(self.scrollable_frame, text="Я администратор",

```

```

                                                                 command=lambda:
self.show_role_instructions("admin"))
    self.admin_button.pack(pady=5)

    self.user_button = tk.Button(self.scrollable_frame, text="Я
обычный пользователь",
                                                                 command=lambda:
self.show_role_instructions("user"))
    self.user_button.pack(pady=5)

    self.building_manager_button = tk.Button(self.scrollable_frame,
text="Я менеджер зданий кинотеатров",
                                                                 command=lambda:
self.show_role_instructions("building_manager"))
    self.building_manager_button.pack(pady=5)

    self.film_manager_button = tk.Button(self.scrollable_frame,
text="Я менеджер фильмотеки",
                                                                 command=lambda:
self.show_role_instructions("film_manager"))
    self.film_manager_button.pack(pady=5)

    self.cinema_director_button = tk.Button(self.scrollable_frame,
text="Я директор кинотеатра",
                                                                 command=lambda:
self.show_role_instructions("cinema_director"))
    self.cinema_director_button.pack(pady=5)

    self.cinema_manager_button = tk.Button(self.scrollable_frame,
text="Я менеджер кинотеатра",
                                                                 command=lambda:
self.show_role_instructions("cinema_manager"))
    self.cinema_manager_button.pack(pady=5)

    self.cashier_button = tk.Button(self.scrollable_frame, text="Я
кассир кинотеатра",
                                                                 command=lambda:
self.show_role_instructions("cashier"))
    self.cashier_button.pack(pady=5)

    self.note_text = tk.Label(self.scrollable_frame, text=(
        "Примечание: Если вы администратор системы и запускаете
клиент в первый раз, "
        "а сервер еще не инициализирован, поле 'Схема' можно
оставить пустым."
    ))
    self.note_text.pack(pady=10)

    self.instruction_text = tk.Label(self.scrollable_frame,
text="", justify="left", wraplength=500)
    self.instruction_text.pack(pady=10)

    self.canvas.bind_all("<MouseWheel>", self._on_mousewheel)

```

```

self.canvas.bind_all("<Button-4>", self._on_mousewheel)
self.canvas.bind_all("<Button-5>", self._on_mousewheel)

def _on_mousewheel(self, event):
    if event.num == 4 or event.delta > 0:
        self.canvas.yview_scroll(-1, "units")
    elif event.num == 5 or event.delta < 0:
        self.canvas.yview_scroll(1, "units")

def show_role_instructions(self, role):
    instructions = {
        "admin": """
Чтобы войти в систему как администратор, авторизируйтесь, используя
логин root
Администратор системы может:
- Создавать новую базу данных для использования в системе
- Указать базу данных, которую нужно использовать в системе
- Создавать пользователей верхнего уровня: менеджера зданий
кинотеатров, менеджера фильмотеки и директора кинотеатра
- Удалять пользователей
Примечание: перед тем, как создавать пользователя-диреткора кинотеатра
убедитесь, что в вашей системе
добавлено хотя бы одно здание кинотеатра, которому можно назначить
директора.

        """,

        "user": """
Чтобы войти в систему как пользователь, авторизируйтесь, используя
Логин user и Пароль user.
Пользователь может:
- Просматривать запланированные в различных кинотеатрах показы сеансов
- Производить поиск сеансов по названию фильма, дате и времени показа,
жанру
- Просматривать информацию о билетах на сеанс, бронировать билеты или
отменять бронь билетов

Примечание: перед тем, как просматривать информацию, бронировать
билеты или отменять бронь, убедитесь,
что вы выбрали кинотеатр и сеанс. Сначала необходимо выбрать
кинотеатр,
а затем сеанс из выпадающих списках.

Чтобы просмотреть информацию о билетах, нажмите соответствующую кнопку
в меню. Перед вами
откроется специальное окно с отображением всех билетов на сеанс.
Зеленый цвет означает, что билет свободен для брони;
Желтый означает, что билет забронирован;
Красный означает, что билет куплен.
Для бронирования билета или отмены брони необходимо ввести номер
телефона.

        """,

        "building_manager": """

```



```

Менеджер зданий кинотеатров может:
- Просматривать информацию о добавленных в систему кинотеатрах
- Добавлять или удалять кинотеатры в системе.
    """

    "film_manager": """
Менеджер фильмотеки может:
- Просматривать информацию о добавленных в фильмотеку фильмах
- Добавлять фильмы в фильмотеку или Удалять их.
    """

    "cinema_director": """
Директор кинотеатра может:
- Просматривать список сотрудников своего кинотеатра
- Нанимать новых сотрудников
- Увольнять сотрудников.
Примечание: у кинотеатра может быть только один директор.
    """

    "cinema_manager": """
Менеджер кинотеатра может:
- Просматривать фильмы, доступные в фильмотеке
- Просматривать расписание кинотеатра (запланированные сеансы)
- Редактировать расписание кинотеатра: создавать сеансы и удалять их,
выбирая фильм из фильмотеки
    """

    "cashier": """
Кассир может:
- просматривать информацию о билетах на выбранный сеанс
- Продавать, бронировать билеты на выбранный сеанс, а также
отменять покупку билета или его бронь

Примечание: Чтобы просмотреть информацию о билетах, выберите сеанс из
выпадающего списка.
Чтобы совершать какие-либо действия с билетами, выберите сеанс и
укажите номер билета.
Чтобы узнать номер билета, откройте специальное окно, нажав на кнопку
"Билеты на сеанс".
    """

    }

    self.instruction_text.config(text=instructions.get(role,
"Инструкция не найдена."))

class RootWindow:
    def __init__(self, master, connection):
        self.master = master
        self.master.title("Окно администратора")
        self.connection = connection

        self.create_db_button = tk.Button(master, text="Создать схему",
command=self.create_database)

```

```

        self.create_db_button.grid(row=0, column=0, padx=10, pady=10)

        self.create_db_button = tk.Button(master, text="Использовать
существующую схему", command=self.use_database)
        self.create_db_button.grid(row=1, column=0, padx=10, pady=10)

        self.create_building_manager_button = tk.Button(master,
text="Создать менеджера культурных строений",
command=self.create_building_manager)
        self.create_building_manager_button.grid(row=0, column=1,
padx=10, pady=10)

        self.create_film_manager_button = tk.Button(master,
text="Создать менеджера фильмотеки", command=self.create_film_manager)
        self.create_film_manager_button.grid(row=1, column=1, padx=10,
pady=10)

        self.create_cinema_director_button = tk.Button(master,
text="Создать директора кинотеатра",
command=self.create_cinema_director)
        self.create_cinema_director_button.grid(row=2, column=1,
padx=10, pady=10)

        self.delete_user_button = tk.Button(master, text="Удалить
пользователя", command=self.delete_user)
        self.delete_user_button.grid(row=0, column=2, padx=10, pady=10)

        self.logout_button = tk.Button(master, text="Выйти",
command=self.logout)
        self.logout_button.grid(row=0, column=3, padx=10, pady=10)

        if 'database' in db_params.keys():
            cursor = self.connection.cursor()
            cursor.execute(f"USE {db_params['database']}")
            connection.commit()

        def use_database(self):
            schema_name = simpledialog.askstring("Input", "Введите имя
схемы:", parent=self.master)

            if schema_name:
                try:
                    cursor = self.connection.cursor()
                    cursor.execute(f"USE {schema_name}")
                    self.connection.database = schema_name # Set the
default schema for the connection
                    messagebox.showinfo("Успех", f"Схема '{schema_name}'
теперь используется по умолчанию.")

                    db_params['database'] = schema_name
                except mysql.connector.Error as err:
                    messagebox.showerror("Ошибка", f"Не удалось
использовать схему '{schema_name}': {err}")

```

```

        finally:
            cursor.close()

    def create_database(self):
        schema_name = simpdialog.askstring("Имя схемы", "Введите имя
схемы:")
        if not schema_name:
            messagebox.showinfo("Error", "Имя схемы не может быть
пустым")
            return

        try:
            cursor = self.connection.cursor()
            cursor.execute(f"CREATE DATABASE IF NOT EXISTS
{schema_name}")
            cursor.execute(f"USE {schema_name}")
            db_params['database'] = schema_name

            # Выполняем скрипты создания таблиц
            self.create_tables(cursor)

            # Выполняем скрипты создания триггеров, процедур и функций
            self.initialize_active_server(cursor)

            # Создаем пользователя user
            self.create_default_user(cursor)

            self.connection.commit()
            messagebox.showinfo("Успех", "База данных и активный сервер
успешно созданы")

        except Error as e:
            messagebox.showinfo("Ошибка", f"Ошибка при создании базы
данных: {e}")

    def create_tables(self, cursor):
        try:
            cursor.execute("""
                CREATE TABLE IF NOT EXISTS cinemas (
                    id INT AUTO_INCREMENT PRIMARY KEY,
                    name VARCHAR(255) NOT NULL,
                    seat_count INT NOT NULL
                )
            """)
            cursor.execute("""
                CREATE TABLE IF NOT EXISTS films (
                    id INT AUTO_INCREMENT PRIMARY KEY,
                    title VARCHAR(255) NOT NULL,
                    genre VARCHAR(255) NOT NULL,
                    duration TIME NOT NULL
                )
            """)
            cursor.execute("""

```

```

        CREATE TABLE IF NOT EXISTS users (
            id INT AUTO_INCREMENT PRIMARY KEY,
            username VARCHAR(255) NOT NULL UNIQUE,
            role ENUM('admin', 'building_manager',
'film_manager', 'cinema_director', 'cinema_manager', 'cashier',
'user') NOT NULL
        )
    """
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS cinema_staff (
            id INT AUTO_INCREMENT PRIMARY KEY,
            username VARCHAR(255) NOT NULL,
            cinema_id INT NOT NULL,
            FOREIGN KEY (username) REFERENCES users(username),
            FOREIGN KEY (cinema_id) REFERENCES cinemas(id)
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS schedule (
            id INT AUTO_INCREMENT PRIMARY KEY,
            cinema_id INT NOT NULL,
            film_id INT NOT NULL,
            showtime DATETIME NOT NULL,
            FOREIGN KEY (cinema_id) REFERENCES cinemas(id),
            FOREIGN KEY (film_id) REFERENCES films(id)
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS tickets (
            id INT AUTO_INCREMENT PRIMARY KEY,
            schedule_id INT NOT NULL,
            seat_number INT NOT NULL,
            status ENUM('free', 'purchased', 'reserved') NOT
NULL,
            FOREIGN KEY (schedule_id) REFERENCES schedule(id)
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS reserved_tickets (
            id INT AUTO_INCREMENT PRIMARY KEY,
            ticket_id INT NOT NULL,
            phone_number VARCHAR(20) NOT NULL,
            FOREIGN KEY (ticket_id) REFERENCES tickets(id)
        )
    """)
except Error as e:
    messagebox.showinfo("Ошибка", f"Ошибка при создании таблиц:
{e}")

def initialize_active_server(self, cursor):
    try:
        # Создание триггеров
        cursor.execute("""

```

```

        CREATE TRIGGER after_insert_schedule
        AFTER INSERT ON schedule
        FOR EACH ROW
        BEGIN
            DECLARE seat_number INT DEFAULT 1;
            DECLARE total_seats INT;
            SELECT seat_count INTO total_seats FROM cinemas
WHERE id = NEW.cinema_id;
            WHILE seat_number <= total_seats DO
                INSERT INTO tickets (schedule_id, seat_number,
status) VALUES (NEW.id, seat_number, 'free');
                SET seat_number = seat_number + 1;
            END WHILE;
        END
    """)
    cursor.execute("""
        CREATE TRIGGER before_delete_schedule
        BEFORE DELETE ON schedule
        FOR EACH ROW
        BEGIN
            DELETE FROM tickets WHERE schedule_id = OLD.id;
        END
    """)
    cursor.execute("""
        CREATE TRIGGER before_delete_tickets
        BEFORE DELETE ON tickets
        FOR EACH ROW
        BEGIN
            DELETE FROM reserved_tickets WHERE ticket_id =
OLD.id;
        END
    """)
    cursor.execute("""
        CREATE TRIGGER before_delete_films
        BEFORE DELETE ON films
        FOR EACH ROW
        BEGIN
            DELETE FROM schedule WHERE film_id = OLD.id;
        END
    """)
    cursor.execute("""
        CREATE TRIGGER before_delete_users
        BEFORE DELETE ON users
        FOR EACH ROW
        BEGIN
            DELETE FROM cinema_staff WHERE username =
OLD.username;
        END
    """)
    cursor.execute("""
        CREATE TRIGGER before_delete_cinemas
        BEFORE DELETE ON cinemas
        FOR EACH ROW
    """)

```

```

        BEGIN
            DECLARE done INT DEFAULT 0;
            DECLARE user_id INT;
            DECLARE user_cursor CURSOR FOR
                SELECT user_id FROM cinema_staff WHERE
cinema_id = OLD.id;
            DECLARE CONTINUE HANDLER FOR NOT FOUND SET done =
1;

            OPEN user_cursor;
            read_loop: LOOP
                FETCH user_cursor INTO user_id;
                IF done THEN
                    LEAVE read_loop;
                END IF;
                CALL delete_user((SELECT username FROM users
WHERE id = user_id), @p_message);
            END LOOP;
            CLOSE user_cursor;
        END
    """)

    # Создание процедур и функций
    cursor.execute("""
        CREATE PROCEDURE sell_ticket(IN p_schedule_id INT, IN
p_ticket_number INT, OUT p_message VARCHAR(255))
        BEGIN
            DECLARE ticket_status VARCHAR(20);
            SELECT status INTO ticket_status
            FROM tickets
            WHERE schedule_id = p_schedule_id AND seat_number =
p_ticket_number;
            IF ticket_status = 'free' THEN
                UPDATE tickets
                SET status = 'purchased'
                WHERE schedule_id = p_schedule_id AND
seat_number = p_ticket_number;
                SET p_message = 'Ticket successfully sold.';
            ELSEIF ticket_status = 'purchased' THEN
                SET p_message = 'Ticket is already sold.';
            ELSEIF ticket_status = 'reserved' THEN
                SET p_message = 'Ticket is reserved.';
            ELSE
                SET p_message = 'Ticket not found.';
            END IF;
        END
    """)

    cursor.execute("""
        CREATE PROCEDURE cancel_ticket(IN p_schedule_id INT, IN
p_ticket_number INT, OUT p_message VARCHAR(255))
        BEGIN
            DECLARE ticket_status VARCHAR(20);
            DECLARE ticket_id INT;
            SELECT status, id INTO ticket_status, ticket_id

```

```

        FROM tickets
        WHERE schedule_id = p_schedule_id AND seat_number =
p_ticket_number;
        IF ticket_status = 'purchased' OR ticket_status =
'reserved' THEN
            IF ticket_status = 'reserved' THEN
                DELETE FROM reserved_tickets WHERE
ticket_id = ticket_id;
            END IF;
            UPDATE tickets
            SET status = 'free'
            WHERE schedule_id = p_schedule_id AND
seat_number = p_ticket_number;
            SET p_message = 'Ticket successfully
canceled.';
        ELSEIF ticket_status = 'free' THEN
            SET p_message = 'Ticket is already free.';
        ELSE
            SET p_message = 'Ticket not found.';
        END IF;
    END
    """)
    cursor.execute("""
        CREATE PROCEDURE confirm_reserve(IN p_schedule_id INT,
IN p_ticket_number INT, IN p_phone_number VARCHAR(20), OUT p_message
VARCHAR(255))
        BEGIN
            DECLARE ticket_status VARCHAR(20);
            DECLARE v_ticket_id INT;

            SELECT status, id INTO ticket_status, v_ticket_id
            FROM tickets
            WHERE schedule_id = p_schedule_id AND seat_number =
p_ticket_number;

            IF ticket_status = 'reserved' THEN
                IF (SELECT phone_number FROM reserved_tickets
rt
                JOIN tickets tck
                ON rt.ticket_id = tck.id
                WHERE tck.schedule_id = p_schedule_id
AND
                tck.seat_number = p_ticket_number)
= p_phone_number THEN
                UPDATE tickets
                SET status = 'purchased'
                WHERE schedule_id = p_schedule_id AND
seat_number = p_ticket_number;

                DELETE FROM reserved_tickets
                WHERE v_ticket_id = ticket_id;

```

```

                SET p_message = 'Ticket reservation
successfully confirmed.';
            ELSE
                SET p_message = 'This ticket reserved by
another phone number';
            END IF;
        ELSEIF ticket_status = 'purchased' THEN
            SET p_message = 'Ticket is already sold.';
        ELSEIF ticket_status = 'free' THEN
            SET p_message = 'This ticket is free.';
        ELSE
            SET p_message = 'Ticket not found.';
        END IF;
    END;
""")
cursor.execute("""
        CREATE PROCEDURE reserve_ticket(IN p_schedule_id INT,
IN p_ticket_number INT, IN p_phone_number VARCHAR(20), OUT p_message
VARCHAR(255))
        BEGIN
            DECLARE ticket_status VARCHAR(20);
            DECLARE ticket_id INT;
            SELECT status, id INTO ticket_status, ticket_id
            FROM tickets
            WHERE schedule_id = p_schedule_id AND seat_number =
p_ticket_number;
            IF ticket_status = 'free' THEN
                UPDATE tickets
                SET status = 'reserved'
                WHERE schedule_id = p_schedule_id AND
seat_number = p_ticket_number;
                INSERT INTO reserved_tickets (ticket_id,
phone_number)
                VALUES (ticket_id, p_phone_number);
                SET p_message = 'Ticket successfully
reserved.';
            ELSEIF ticket_status = 'purchased' THEN
                SET p_message = 'Ticket is already sold.';
            ELSEIF ticket_status = 'reserved' THEN
                SET p_message = 'Ticket is already reserved.';
            ELSE
                SET p_message = 'Ticket not found.';
            END IF;
        END
""")
cursor.execute("""
        CREATE PROCEDURE create_building_manager(IN p_login
VARCHAR(255), IN p_password VARCHAR(255), OUT p_message VARCHAR(255))
        BEGIN
            DECLARE user_count INT;
            DECLARE current_host VARCHAR(255);
            DECLARE create_user_query VARCHAR(500);
            DECLARE grant_privileges_query VARCHAR(500);

```



```

-- Получаем хост текущего пользователя
SET current_host = SUBSTRING_INDEX(USER(), '@', -1);

-- Проверка, существует ли пользователь с таким логином
SELECT COUNT(*) INTO user_count FROM mysql.user WHERE user =
p_login AND host = current_host;

IF user_count = 0 THEN
    -- Создание пользователя в MySQL
    SET @create_user_query = CONCAT('CREATE USER \\'', p_login,
    '\\\'@\\\'', current_host, '\\\' IDENTIFIED BY \\'', p_password, '\\\'');
    PREPARE stmt FROM @create_user_query;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;

    -- Назначение привилегий
    SET @grant_privileges_query = CONCAT('GRANT SELECT, INSERT,
DELETE ON cinemas TO \\'', p_login, '\\\'@\\\'', current_host, '\\\'');
    PREPARE stmt FROM @grant_privileges_query;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;

    SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON FUNCTION
get_role TO \\'', p_login, '\\\'@\\\'', current_host, '\\\'');
    PREPARE stmt FROM @grant_privileges_query;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;

    -- Вставка пользователя в таблицу users
    INSERT INTO users (username, role) VALUES (p_login,
'building_manager');

    SET p_message = 'Building manager created successfully.';
ELSE
    SET p_message = 'User with this login already exists.';
END IF;
END;

""")
cursor.execute("""
CREATE PROCEDURE create_film_manager(IN p_login
VARCHAR(255), IN p_password VARCHAR(255), OUT p_message VARCHAR(255))
BEGIN
    DECLARE user_count INT;
    DECLARE current_host VARCHAR(255);
    DECLARE create_user_query VARCHAR(500);
    DECLARE grant_privileges_query VARCHAR(500);

    -- Получаем хост текущего пользователя
    SET current_host = SUBSTRING_INDEX(USER(), '@', -1);

    -- Проверка, существует ли пользователь с таким логином

```

```

        SELECT COUNT(*) INTO user_count FROM mysql.user WHERE user =
p_login AND host = current_host;

    IF user_count = 0 THEN
        -- Создание пользователя в MySQL
        SET @create_user_query = CONCAT('CREATE USER \\'', p_login,
'\\'@\\', current_host, '\\' IDENTIFIED BY \\'', p_password, '\\';');
        PREPARE stmt FROM @create_user_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;

        -- Назначение привилегий
        SET @grant_privileges_query = CONCAT('GRANT SELECT, INSERT,
DELETE ON films TO \\'', p_login, '\\'@\\', current_host, '\\';');
        PREPARE stmt FROM @grant_privileges_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;

        SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON FUNCTION
get_role TO \\'', p_login, '\\'@\\', current_host, '\\';');
        PREPARE stmt FROM @grant_privileges_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;

        -- Вставка пользователя в таблицу users
        INSERT INTO users (username, role) VALUES (p_login,
'film_manager');

        SET p_message = 'Film manager created successfully.';
    ELSE
        SET p_message = 'User with this login already exists.';
    END IF;
END;

""")
cursor.execute("""
        CREATE PROCEDURE create_cinema_director(IN p_login
VARCHAR(255), IN p_password VARCHAR(255), IN p_target_cinema_id INT,
OUT p_message VARCHAR(255))
BEGIN
    DECLARE director_count INT;
    DECLARE user_count INT;
    DECLARE current_host VARCHAR(255);
    DECLARE create_user_query VARCHAR(500);
    DECLARE grant_privileges_query VARCHAR(500);

    -- Получаем хост текущего пользователя
    SET current_host = SUBSTRING_INDEX(USER(), '@', -1);

    -- Проверка, существует ли директор для данного кинотеатра
    SELECT COUNT(*) INTO director_count
    FROM cinema_staff cs
    JOIN users u ON cs.username = u.username

```

```

        WHERE cs.cinema_id = p_target_cinema_id AND u.role =
'cinema_director';

    IF director_count = 0 THEN
        -- Проверка, существует ли пользователь с таким логином
        SELECT COUNT(*) INTO user_count FROM mysql.user WHERE user =
p_login AND host = current_host;

        IF user_count = 0 THEN
            -- Создание пользователя в MySQL
            SET @create_user_query = CONCAT('CREATE USER \\'', p_login,
'\\'@\\', current_host, '\\' IDENTIFIED BY \\'', p_password, '\\';');
            PREPARE stmt FROM @create_user_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

            -- Назначение привилегий
            SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE see_cinema_staff TO \\'', p_login, '\\'@\\', current_host,
'\\';');
            PREPARE stmt FROM @grant_privileges_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

            SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE add_staff TO \\'', p_login, '\\'@\\', current_host,
'\\';');
            PREPARE stmt FROM @grant_privileges_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

            SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE delete_staff TO \\'', p_login, '\\'@\\', current_host,
'\\';');
            PREPARE stmt FROM @grant_privileges_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

            SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
FUNCTION get_cinema_id TO \\'', p_login, '\\'@\\', current_host,
'\\';');
            PREPARE stmt FROM @grant_privileges_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

            SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
FUNCTION get_role TO \\'', p_login, '\\'@\\', current_host, '\\';');
            PREPARE stmt FROM @grant_privileges_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

            -- Вставка пользователя в таблицу users

```

```

        INSERT INTO users (username, role) VALUES (p_login,
'cinema_director');
        -- Вставка записи в таблицу cinema_staff
        INSERT INTO cinema_staff (username, cinema_id)
VALUES (p_login, p_target_cinema_id);

        SET p_message = 'Cinema director created successfully.';
    ELSE
        SET p_message = 'User with this login already exists.';
    END IF;
ELSE
    SET p_message = 'This cinema already has a director.';
END IF;
END;
""")
cursor.execute("""
        CREATE PROCEDURE see_cinema_staff(IN p_target_cinema_id
INT)
        BEGIN
            SELECT u.username, u.role
            FROM cinema_staff cs
            JOIN users u ON cs.username = u.username
            WHERE cs.cinema_id = p_target_cinema_id;
        END;
""")
cursor.execute("""
        CREATE PROCEDURE add_staff(IN p_login
VARCHAR(255), IN p_password VARCHAR(255), IN p_role VARCHAR(255), OUT
p_message VARCHAR(255))
        BEGIN
            DECLARE user_count INT;
            DECLARE current_host VARCHAR(255);
            DECLARE create_user_query VARCHAR(500);
            DECLARE grant_privileges_query VARCHAR(500);
            DECLARE cinema_id INT;

            -- Получаем хост текущего пользователя
            SET current_host = SUBSTRING_INDEX(USER(), '@', -1);

            -- Проверка, существует ли пользователь с таким логином
            SELECT COUNT(*) INTO user_count FROM mysql.user WHERE user =
p_login AND host = current_host;

            IF user_count = 0 THEN
                -- Получение cinema_id для данного пользователя
                SET cinema_id = get_cinema_id();

                -- Проверка роли
                IF p_role NOT IN ('cinema_manager', 'cashier') THEN
                    SET p_message = CONCAT('There is no such position in the
cinema: ', p_role);
                ELSE
                    -- Создание пользователя в MySQL

```

```

        SET @create_user_query = CONCAT('CREATE USER \\\', p_login,
'\\'@\\', current_host, '\\\' IDENTIFIED BY \\\', p_password, '\\';');
        PREPARE stmt FROM @create_user_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;

        -- Назначение привилегий
        IF p_role = 'cinema_manager' THEN
            SET @grant_privileges_query = CONCAT('GRANT SELECT ON
films TO \\\', p_login, '\\\'@\\', current_host, '\\';');
            PREPARE stmt FROM @grant_privileges_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

            SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE see_schedule TO \\\', p_login, '\\\'@\\', current_host,
'\\';');
            PREPARE stmt FROM @grant_privileges_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

            SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE add_schedule TO \\\', p_login, '\\\'@\\', current_host,
'\\';');
            PREPARE stmt FROM @grant_privileges_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

            SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE delete_schedule TO \\\', p_login, '\\\'@\\', current_host,
'\\';');
            PREPARE stmt FROM @grant_privileges_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

            SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
FUNCTION get_cinema_id TO \\\', p_login, '\\\'@\\', current_host,
'\\';');
            PREPARE stmt FROM @grant_privileges_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

            SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
FUNCTION get_role TO \\\', p_login, '\\\'@\\', current_host, '\\';');
            PREPARE stmt FROM @grant_privileges_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

        ELSEIF p_role = 'cashier' THEN
            SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE sell_ticket TO \\\', p_login, '\\\'@\\', current_host,
'\\';');
            PREPARE stmt FROM @grant_privileges_query;

```

```

EXECUTE stmt;
DEALLOCATE PREPARE stmt;

SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE cancel_ticket TO \\', p_login, '\\\\'@\\\\', current_host,
'\\\\;');
PREPARE stmt FROM @grant_privileges_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE reserve_ticket TO \\', p_login, '\\\\'@\\\\', current_host,
'\\\\;');
PREPARE stmt FROM @grant_privileges_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE confirm_reserve TO \\', p_login, '\\\\'@\\\\', current_host,
'\\\\;');
PREPARE stmt FROM @grant_privileges_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
PROCEDURE see_schedule TO \\', p_login, '\\\\'@\\\\', current_host,
'\\\\;');
PREPARE stmt FROM @grant_privileges_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
FUNCTION get_cinema_id TO \\', p_login, '\\\\'@\\\\', current_host,
'\\\\;');
PREPARE stmt FROM @grant_privileges_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
FUNCTION count_free_seats TO \\', p_login, '\\\\'@\\\\', current_host,
'\\\\;');
PREPARE stmt FROM @grant_privileges_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
FUNCTION count_occupied_tickets TO \\', p_login, '\\\\'@\\\\',
current_host, '\\\\;');
PREPARE stmt FROM @grant_privileges_query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

```

```

        SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
FUNCTION get_ticket_status TO \\'', p_login, '\\\'@\\\'', current_host,
'\\\'');
        PREPARE stmt FROM @grant_privileges_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;

        SET @grant_privileges_query = CONCAT('GRANT EXECUTE ON
FUNCTION get_role TO \\'', p_login, '\\\'@\\\'', current_host, '\\\'');
        PREPARE stmt FROM @grant_privileges_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;
    END IF;

    -- Вставка пользователя в таблицу users
    INSERT INTO users (username, role) VALUES (p_login,
p_role);

    -- Вставка записи в таблицу cinema_staff
    INSERT INTO cinema_staff (username, cinema_id)
VALUES (p_login, cinema_id);

    SET p_message = 'Staff member added successfully.';
END IF;
ELSE
    SET p_message = 'User with this login already exists.';
END IF;
END;

""")
cursor.execute("""
CREATE PROCEDURE delete_staff(IN p_login VARCHAR(255), OUT
p_message VARCHAR(255))
BEGIN
    DECLARE user_count INT;
    DECLARE cinema_id INT;
    DECLARE current_host VARCHAR(255);
    DECLARE drop_user_query VARCHAR(500);

    -- Получаем хост текущего пользователя
    SET current_host = SUBSTRING_INDEX(USER(), '@', -1);

    -- Проверка, существует ли пользователь с таким логином
    SELECT COUNT(*) INTO user_count FROM mysql.user WHERE
user = p_login AND host = current_host;

    IF user_count = 0 THEN
        SET p_message = 'User with this login does not
exist.';
    ELSE
        -- Проверка, является ли пользователь сотрудником
данного кинотеатра
        SELECT cs.cinema_id INTO cinema_id
        FROM cinema_staff cs
        WHERE cs.username = p_login;
    
```

```

        IF cinema_id != (SELECT cs.cinema_id
                        FROM cinema_staff cs
                        WHERE cs.username =
SUBSTRING_INDEX(USER(), '@', 1)) THEN
        SET p_message = 'This user does not belong to
your cinema.';
    ELSE
        -- Удаление пользователя из MySQL
        SET @drop_user_query = CONCAT('DROP USER \\'',
p_login, '\\\'@\\\'', current_host, '\\\'');
        PREPARE stmt FROM @drop_user_query;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;

        -- Удаление записи из таблиц users и
cinema_staff
        DELETE FROM users WHERE username = p_login;
        DELETE FROM cinema_staff WHERE username =
p_login;

        SET p_message = 'Staff member deleted
successfully.';
    END IF;
END IF;
END;
""")
cursor.execute("""
CREATE PROCEDURE add_schedule(IN p_target_film_id INT, IN
p_target_showtime DATETIME, OUT p_message VARCHAR(255))
BEGIN
    DECLARE cinema_id INT;

    -- Получение cinema_id для данного пользователя
    SET cinema_id = get_cinema_id();

    -- Вставка новой записи в расписание
    INSERT INTO schedule (cinema_id, film_id, showtime)
VALUES (cinema_id, p_target_film_id,
p_target_showtime);

    SET p_message = 'Schedule added successfully.';
END;
""")
cursor.execute("""
CREATE PROCEDURE delete_schedule(IN p_schedule_id INT, OUT
p_message VARCHAR(255))
BEGIN
    DECLARE schedule_cinema_id INT;
    DECLARE user_cinema_id INT;

    -- Получение cinema_id для данной записи расписания

```



```

        SELECT cinema_id INTO schedule_cinema_id FROM schedule
WHERE id = p_schedule_id;

        -- Получение cinema_id текущего пользователя
        SELECT cs.cinema_id INTO user_cinema_id
        FROM cinema_staff cs
        WHERE cs.username = SUBSTRING_INDEX(USER(), '@', 1);

        IF schedule_cinema_id = user_cinema_id THEN
            -- Удаление записи из расписания
            DELETE FROM schedule WHERE id = p_schedule_id;
            SET p_message = 'Schedule deleted successfully.';
        ELSE
            SET p_message = 'Schedule not found or you do not
have permission to delete it.';
        END IF;
    END;
    """
    cursor.execute("""
CREATE PROCEDURE see_schedule()
BEGIN
        SELECT s.id, c.name AS cinema_name, f.title AS
film_title, s.showtime
        FROM schedule s
        JOIN cinemas c ON s.cinema_id = c.id
        JOIN films f ON s.film_id = f.id
        WHERE s.cinema_id = (SELECT cs.cinema_id
                            FROM cinema_staff cs
                            WHERE cs.username =
SUBSTRING_INDEX(USER(), '@', 1));
    END;
    """)
    cursor.execute("""
CREATE PROCEDURE see_tickets()
BEGIN
        SELECT t.id, t.schedule_id, t.seat_number, t.status
        FROM tickets t
        JOIN schedule s ON t.schedule_id = s.id
        WHERE s.cinema_id = (SELECT cs.cinema_id
                            FROM cinema_staff cs
                            WHERE cs.username =
SUBSTRING_INDEX(USER(), '@', 1));
    END;
    """)
    cursor.execute("""
CREATE PROCEDURE delete_user(IN p_login VARCHAR(255), OUT
p_message VARCHAR(255))
BEGIN
        DECLARE user_id INT;
        DECLARE user_role ENUM('admin', 'building_manager',
'film_manager', 'cinema_director', 'cinema_manager', 'cashier',
'user');
        DECLARE current_host VARCHAR(255);

```

```

        DECLARE drop_user_query VARCHAR(500);

        -- Получаем хост текущего пользователя
        SET current_host = SUBSTRING_INDEX(USER(), '@', -1);

        -- Проверка, существует ли пользователь с таким логином
        SELECT id, role INTO user_id, user_role FROM users
WHERE username = p_login;

        IF user_id IS NULL THEN
            SET p_message = 'User with this login does not
exist.';
        ELSE
            -- Проверка, является ли пользователь кассиром,
            директором или менеджером кинотеатра
            IF user_role IN ('cinema_director',
'cinema_manager', 'cashier') THEN
                -- Удаление записи из таблицы cinema_staff
                DELETE FROM cinema_staff WHERE username =
p_login;
            END IF;

            -- Удаление записи из таблицы users
            DELETE FROM users WHERE username = p_login;

            -- Удаление пользователя из MySQL
            SET @drop_user_query = CONCAT('DROP USER \\'',
p_login, '\\\'@\\\'', current_host, '\\\'');
            PREPARE stmt FROM @drop_user_query;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;

            SET p_message = 'User deleted successfully.';
        END IF;
    END;
    """)

# Создание функций
cursor.execute("""
        CREATE FUNCTION count_free_seats(p_schedule_id INT)
RETURNS INT
        READS SQL DATA
        BEGIN
            DECLARE free_seats INT;
            SELECT COUNT(*) INTO free_seats
            FROM tickets
            WHERE schedule_id = p_schedule_id AND status =
'free';
            RETURN free_seats;
        END
    """)
    cursor.execute("""

```

```

        CREATE FUNCTION count_occupied_tickets(p_schedule_id
INT) RETURNS INT
        READS SQL DATA
        BEGIN
            DECLARE occupied_tickets INT;
            SELECT COUNT(*) INTO occupied_tickets
            FROM tickets
            WHERE schedule_id = p_schedule_id AND status IN
('purchased', 'reserved');
            RETURN occupied_tickets;
        END
    """)
    cursor.execute("""
        CREATE FUNCTION get_ticket_status(p_schedule_id INT,
p_ticket_number INT) RETURNS VARCHAR(20)
        READS SQL DATA
        BEGIN
            DECLARE ticket_status VARCHAR(20);
            SELECT status INTO ticket_status
            FROM tickets
            WHERE schedule_id = p_schedule_id AND seat_number =
p_ticket_number;
            IF ticket_status IS NULL THEN
                RETURN 'Ticket not found';
            ELSE
                RETURN ticket_status;
            END IF;
        END
    """)
    cursor.execute("""
        CREATE FUNCTION get_cinema_id() RETURNS INT
        READS SQL DATA
        BEGIN
            RETURN (SELECT cinema_id
                    FROM cinema_staff cs
                    WHERE cs.username = SUBSTRING_INDEX(USER(),
'@', 1));
        END
    """)
    cursor.execute("""
        CREATE FUNCTION get_role() RETURNS VARCHAR(255)
        READS SQL DATA
        BEGIN
            RETURN (SELECT role
                    FROM users us
                    WHERE us.username = SUBSTRING_INDEX(USER(),
'@', 1));
        END;
    """)

except Error as e:
    messagebox.showinfo("Ошибка", f"Ошибка при инициализации
активного сервера: {e}")

```

```

def create_default_user(self, cursor):
    try:
        cursor.execute("""
            CREATE USER IF NOT EXISTS 'user'@'localhost' IDENTIFIED
BY 'user';
        """)
        cursor.execute("""
            GRANT SELECT ON schedule TO 'user'@'localhost';
        """)
        cursor.execute("""
            GRANT SELECT ON cinemas TO 'user'@'localhost';
        """)
        cursor.execute("""
            GRANT SELECT ON films TO 'user'@'localhost';
        """)
        cursor.execute("""
            GRANT EXECUTE ON PROCEDURE reserve_ticket TO
'user'@'localhost';
        """)
        cursor.execute("""
            GRANT EXECUTE ON PROCEDURE cancel_ticket TO
'user'@'localhost';
        """)
        cursor.execute("""
            GRANT EXECUTE ON FUNCTION count_free_seats TO
'user'@'localhost';
        """)
        cursor.execute("""
            GRANT EXECUTE ON FUNCTION count_occupied_tickets TO
'user'@'localhost';
        """)
        cursor.execute("""
            GRANT EXECUTE ON FUNCTION get_ticket_status TO
'user'@'localhost';
        """)
        cursor.execute("""
            GRANT EXECUTE ON FUNCTION get_role TO
'user'@'localhost';
        """)
        cursor.execute("""
            INSERT INTO users (username, role) VALUES ('user',
'user')
            ON DUPLICATE KEY UPDATE role = 'user';
        """)
    except Error as e:
        messagebox.showinfo("Ошибка", f"Ошибка при создании
пользователя 'user': {e}")

    def create_building_manager(self):
        self.create_user_window('Создать менеджера культурных
строений', 'create_building_manager')

```

```

def create_film_manager(self):
    self.create_user_window('Создать менеджера фильмотеки',
'create_film_manager')

def create_cinema_director(self):
    window = tk.Toplevel(self.master)
    window.title('Создать директора кинотеатра')

    tk.Label(window, text='Логин:').grid(row=0, column=0)
    login_entry = tk.Entry(window)
    login_entry.grid(row=0, column=1)

    tk.Label(window, text='Пароль:').grid(row=1, column=0)
    password_entry = tk.Entry(window, show="*")
    password_entry.grid(row=1, column=1)

    tk.Label(window, text='Кинотеатр:').grid(row=2, column=0)
    cinema_combobox = ttk.Combobox(window)
    cinema_combobox.grid(row=2, column=1)

    try:
        cursor = self.connection.cursor()
        cursor.execute("SELECT id, name FROM cinemas")
        cinemas = cursor.fetchall()
        cinema_combobox['values'] = [f"{cinema[0]}: {cinema[1]}"
for cinema in cinemas]
    except Error as e:
        messagebox.showinfo("Ошибка", f"Ошибка при получении списка
кинотеатров: {e}")

    def on_submit():
        login = login_entry.get()
        password = password_entry.get()
        cinema = cinema_combobox.get()

        if not login or not password or not cinema:
            messagebox.showinfo("Ошибка", "Все поля должны быть
заполнены")

            return

        cinema_id = cinema.split(":")[0]

        try:
            p_message = ''
            result_args = cursor.callproc('create_cinema_director',
[login, password, cinema_id, p_message])
            p_message = result_args[3]
            messagebox.showinfo("Сообщение", p_message)
            self.connection.commit()
        except Error as e:
            messagebox.showinfo("Ошибка", f"Ошибка при создании
директора кинотеатра: {e}")
            window.destroy()

```

```

submit_button = tk.Button(window, text="OK", command=on_submit)
submit_button.grid(row=3, columnspan=2)

def create_user_window(self, title, procedure_name):
    window = tk.Toplevel(self.master)
    window.title(title)

    tk.Label(window, text='Логин:').grid(row=0, column=0)
    login_entry = tk.Entry(window)
    login_entry.grid(row=0, column=1)

    tk.Label(window, text='Пароль:').grid(row=1, column=0)
    password_entry = tk.Entry(window, show="*")
    password_entry.grid(row=1, column=1)

    def on_submit():
        login = login_entry.get()
        password = password_entry.get()

        if not login or not password:
            messagebox.showinfo("Ошибка", "Все поля должны быть
заполнены")
            return

        try:
            cursor = self.connection.cursor()
            p_message = ''
            result_args = cursor.callproc(procedure_name, [login,
password, p_message])
            p_message = result_args[2]
            messagebox.showinfo("Сообщение", p_message)
            self.connection.commit()
        except Error as e:
            messagebox.showinfo("Ошибка", f"Ошибка при создании
пользователя: {e}")
            window.destroy()

    submit_button = tk.Button(window, text="OK", command=on_submit)
    submit_button.grid(row=2, columnspan=2)

def delete_user(self):
    window = tk.Toplevel(self.master)
    window.title('Удалить пользователя')

    tk.Label(window, text='Логин:').grid(row=0, column=0)
    login_entry = tk.Entry(window)
    login_entry.grid(row=0, column=1)

    def on_submit():
        login = login_entry.get()

        if not login:

```

```

        messagebox.showinfo("Ошибка", "Логин не может быть
пустым")

        return

    try:
        cursor = self.connection.cursor()
        cursor.execute(f"SET @p_msg='';")
        cursor.execute(f"CALL delete_user('{login}', @p_msg);")
        cursor.execute("SELECT @p_msg;")
        p_message = cursor.fetchone()[0]
        messagebox.showinfo("Сообщение", p_message)
        self.connection.commit()
    except Error as e:
        messagebox.showinfo("Ошибка", f"Ошибка при удалении
пользователя: {e}")
    window.destroy()

    submit_button = tk.Button(window, text="OK", command=on_submit)
    submit_button.grid(row=1, columnspan=2)

def logout(self):
    self.master.destroy()
    login_window = tk.Tk()
    default_font = tkfont.Font(family="Helvetica", size=font_size,
weight="normal")
    login_window.option_add("*Font", default_font)
    self.connection.close()
    LoginWindow(login_window)
    login_window.mainloop()

class BuildingManagerWindow:
    def __init__(self, master, connection):
        self.master = master
        self.connection = connection
        self.master.title("Менеджер зданий кинотеатров")

        self.tree = ttk.Treeview(master, columns=("ID", "Name",
"Seats"), show='headings')
        self.tree.heading("ID", text="Номер кинотеатра")
        self.tree.heading("Name", text="Название")
        self.tree.heading("Seats", text="Количество свободных мест")
        self.tree.grid(row=0, columnspan=3)

        self.add_button = tk.Button(master, text="Добавить кинотеатр",
command=self.add_building)
        self.add_button.grid(row=1, column=0)

        self.delete_button = tk.Button(master, text="Удалить
кинотеатр", command=self.delete_building)
        self.delete_button.grid(row=1, column=1)

```

```

        self.logout_button = tk.Button(master, text="Выйти",
command=self.logout)
        self.logout_button.grid(row=1, column=2)

        self.load_cinemas()

def load_cinemas(self):
    for row in self.tree.get_children():
        self.tree.delete(row)

    try:
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM cinemas")
        rows = cursor.fetchall()

        for row in rows:
            self.tree.insert("", tk.END, values=row)

        cursor.close()
    except Error as e:
        messagebox.showerror("Ошибка", f"Ошибка при получении
данных с сервера: {e}")

def add_building(self):
    add_window = tk.Toplevel(self.master)
    add_window.title("Добавить кинотеатр")

    tk.Label(add_window, text="Название нового
кинотеатра:").grid(row=0, column=0)
    name_entry = tk.Entry(add_window)
    name_entry.grid(row=0, column=1)

    tk.Label(add_window, text="Количество мест в
зале:").grid(row=1, column=0)
    seats_entry = tk.Entry(add_window)
    seats_entry.grid(row=1, column=1)

    def on_add():
        name = name_entry.get()
        seats = seats_entry.get()

        if not name or not seats:
            messagebox.showinfo("Ошибка", "Все поля должны быть
заполнены")

            return

        try:
            cursor = self.connection.cursor()
            cursor.execute("INSERT INTO cinemas (name, seat_count)
VALUES (%s, %s)", (name, seats))
            self.connection.commit()
            cursor.close()
            self.load_cinemas()

```



```

        add_window.destroy()
        messagebox.showinfo("Успех", "Кинотеатр успешно
добавлен")
    except Error as e:
        messagebox.showerror("Ошибка", f"Ошибка при добавлении
кинотеатра: {e}")

    tk.Button(add_window, text="OK", command=on_add).grid(row=2,
columnspan=2)

    def delete_building(self):
        delete_window = tk.Toplevel(self.master)
        delete_window.title("Удалить кинотеатр")

        tk.Label(delete_window, text="Выберите кинотеатр, который
хотите удалить:").grid(row=0, column=0)
        building_id_combobox = ttk.Combobox(delete_window)
        building_id_combobox.grid(row=0, column=1)

        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT id, name FROM cinemas")
            buildings = cursor.fetchall()
            building_id_combobox['values'] = [f"{row[0]}: {row[1]}" for
row in buildings]
            cursor.close()
        except Error as e:
            messagebox.showerror("Ошибка", f"Ошибка при получении
данных: {e}")

    def on_delete():
        selected = building_id_combobox.get()
        if not selected:
            messagebox.showinfo("Ошибка", "Не выбран кинотеатр,
который нужно удалить")
            return

        building_id = int(selected.split(":")[0])

        try:
            cursor = self.connection.cursor()
            cursor.execute("DELETE FROM cinemas WHERE id = %s",
(building_id,))
            self.connection.commit()
            cursor.close()
            self.load_cinemas()
            delete_window.destroy()
            messagebox.showinfo("Успех", "Кинотеатр удален")
        except Error as e:
            messagebox.showerror("Ошибка", f"Ошибка при удалении
кинотеатра: {e}")

```

```

tk.Button(delete_window, text="OK",
command=on_delete).grid(row=1, columnspan=2)

def logout(self):
    self.master.destroy()
    root = tk.Tk()
    default_font = tkfont.Font(family="Helvetica", size=font_size,
weight="normal")
    root.option_add("*Font", default_font)
    self.connection.close()
    LoginWindow(root)
    root.mainloop()

class FilmManagerWindow:
    def __init__(self, master, connection):
        self.master = master
        self.connection = connection
        self.master.title("менеджер фильмотеки")

        self.tree = ttk.Treeview(master, columns=("ID", "Title",
"Genre", "Duration"), show='headings')
        self.tree.heading("ID", text="Номер фильма")
        self.tree.heading("Title", text="Название")
        self.tree.heading("Genre", text="Жанр")
        self.tree.heading("Duration", text="Длительность")
        self.tree.grid(row=0, columnspan=3)

        self.add_button = tk.Button(master, text="Добавить новый
фильм", command=self.add_film)
        self.add_button.grid(row=1, column=0)

        self.delete_button = tk.Button(master, text="Удалить фильм",
command=self.delete_film)
        self.delete_button.grid(row=1, column=1)

        self.logout_button = tk.Button(master, text="Выйти",
command=self.logout)
        self.logout_button.grid(row=1, column=2)

        self.load_films()

    def load_films(self):
        for row in self.tree.get_children():
            self.tree.delete(row)

        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM films")
            rows = cursor.fetchall()

            for row in rows:
                self.tree.insert("", tk.END, values=row)

```

```

        cursor.close()
    except Error as e:
        messagebox.showerror("Ошибка", f"Ошибка при получении
данных: {e}")

    def add_film(self):
        add_window = tk.Toplevel(self.master)
        add_window.title("Добавить фильм")

        tk.Label(add_window, text="Название:").grid(row=0, column=0)
        title_entry = tk.Entry(add_window)
        title_entry.grid(row=0, column=1)

        tk.Label(add_window, text="Жанр:").grid(row=1, column=0)
        genre_entry = tk.Entry(add_window)
        genre_entry.grid(row=1, column=1)

        tk.Label(add_window, text="Длительность
(HH:MM:SS):").grid(row=2, column=0)
        duration_entry = tk.Entry(add_window)
        duration_entry.grid(row=2, column=1)

    def on_add():
        title = title_entry.get()
        genre = genre_entry.get()
        duration = duration_entry.get()

        if not title or not genre or not duration:
            messagebox.showinfo("Ошибка", "Все поля должны быть
заполнены")

            return

        try:
            cursor = self.connection.cursor()
            cursor.execute("INSERT INTO films (title, genre,
duration) VALUES (%s, %s, %s)", (title, genre, duration))
            self.connection.commit()
            cursor.close()
            self.load_films()
            add_window.destroy()
            messagebox.showinfo("Успех", "Фильм успешно добавлен")
        except Error as e:
            messagebox.showerror("Ошибка", f"Ошибка при добавлении
фильма: {e}")

        tk.Button(add_window, text="OK", command=on_add).grid(row=3,
columnspan=2)

    def delete_film(self):
        delete_window = tk.Toplevel(self.master)
        delete_window.title("Удалить фильм")

```

```

        tk.Label(delete_window, text="Выберите фильм:").grid(row=0,
column=0)
        film_id_combobox = ttk.Combobox(delete_window)
        film_id_combobox.grid(row=0, column=1)

        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT id, title FROM films")
            films = cursor.fetchall()
            film_id_combobox['values'] = [f"{row[0]}: {row[1]}" for row
in films]
            cursor.close()
        except Error as e:
            messagebox.showerror("Ошибка", f"Ошибка при получении
данных: {e}")

        def on_delete():
            selected = film_id_combobox.get()
            if not selected:
                messagebox.showinfo("Ошибка", "Нужно выбрать фильм")
                return

            film_id = int(selected.split(":")[0])

            try:
                cursor = self.connection.cursor()
                cursor.execute("DELETE FROM films WHERE id = %s",
(film_id,))
                self.connection.commit()
                cursor.close()
                self.load_films()
                delete_window.destroy()
                messagebox.showinfo("Успех", "Фильм удален")
            except Error as e:
                messagebox.showerror("Error", f"Ошибка при удалении
фильма: {e}")

            tk.Button(delete_window, text="OK",
command=on_delete).grid(row=1, columnspan=2)

        def logout(self):
            self.master.destroy()
            root = tk.Tk()
            default_font = tkfont.Font(family="Helvetica", size=font_size,
weight="normal")
            root.option_add("*Font", default_font)
            self.connection.close()
            LoginWindow(root)
            root.mainloop()

class CinemaDirectorWindow:
    def __init__(self, master, connection):

```

```

self.master = master
self.connection = connection
self.master.title("Директор кинотеатра")

        self.view_staff_button = tk.Button(master, text="Список
сотрудников", command=self.view_staff)
        self.view_staff_button.grid(row=0, column=0)

        self.add_staff_button = tk.Button(master, text="Добавить
сотрудника", command=self.add_staff)
        self.add_staff_button.grid(row=0, column=1)

        self.delete_staff_button = tk.Button(master, text="Удалить
сотрудника", command=self.delete_staff)
        self.delete_staff_button.grid(row=0, column=2)

        self.logout_button = tk.Button(master, text="Выйти",
command=self.logout)
        self.logout_button.grid(row=0, column=3)

def get_cinema_id(self):
    try:
        cursor = self.connection.cursor()
        cursor.execute("SELECT get_cinema_id()")
        cinema_id = cursor.fetchone()[0]
        cursor.close()
        return cinema_id
    except Error as e:
        messagebox.showerror("Ошибка", f"Ошибка при получении
данных о кинотеатре: {e}")
        return None

def view_staff(self):
    cinema_id = self.get_cinema_id()
    if cinema_id is None:
        return

    view_window = tk.Toplevel(self.master)
    view_window.title("Работники кинотеатра")

    tree = ttk.Treeview(view_window, columns=("Username", "Role"),
show='headings')
    tree.heading("Username", text="Имя пользователя")
    tree.heading("Role", text="Должность")
    tree.grid(row=0, column=0, columnspan=2)

    try:
        cursor = self.connection.cursor()
        cursor.callproc('see_cinema_staff', [cinema_id])
        for result in cursor.stored_results():
            rows = result.fetchall()
            for row in rows:
                if row[1] == 'cinema_director':

```

```

        role = 'Директор'
    elif row[1] == 'cashier':
        role = 'Кассир'
    elif row[1] == 'cinema_manager':
        role = 'Менеджер'
    else:
        role = 'Неизвестно'
    tree.insert("", tk.END, values=(row[0], role))
    cursor.close()
except Error as e:
    messagebox.showerror("Ошибка", f"Ошибка при получении
данных: {e}")

def add_staff(self):
    add_window = tk.Toplevel(self.master)
    add_window.title("Добавить сотрудника")

    tk.Label(add_window, text="Логин:").grid(row=0, column=0)
    login_entry = tk.Entry(add_window)
    login_entry.grid(row=0, column=1)

    tk.Label(add_window, text="Пароль:").grid(row=1, column=0)
    password_entry = tk.Entry(add_window, show="*")
    password_entry.grid(row=1, column=1)

    tk.Label(add_window, text="Должность:").grid(row=2, column=0)
    role_combobox = ttk.Combobox(add_window, values=["Менеджер",
"Кассир"])
    role_combobox.grid(row=2, column=1)

    def on_add():
        login = login_entry.get()
        password = password_entry.get()
        role = role_combobox.get()
        if role == "Менеджер":
            role = 'cinema_manager'
        elif role == "Кассир":
            role = 'cashier'

        if not login or not password or not role:
            messagebox.showinfo("Ошибка", "Все поля должны быть
заполнены")

        return

    try:
        cursor = self.connection.cursor()
        message = ''
        res = cursor.callproc('add_staff', [login, password,
role, message])
        message = res[3]
        cursor.close()
        self.connection.commit()
        add_window.destroy()

```

```

        messagebox.showinfo("Успех", message)
    except Error as e:
        messagebox.showerror("Ошибка", f"Ошибка при добавлении
сотрудника: {e}")

    tk.Button(add_window, text="OK", command=on_add).grid(row=3,
columnspan=2)

    def delete_staff(self):
        delete_window = tk.Toplevel(self.master)
        delete_window.title("Удаление сотрудника")

        tk.Label(delete_window, text="Введите логин
сотрудника:").grid(row=0, column=0)
        login_entry = tk.Entry(delete_window)
        login_entry.grid(row=0, column=1)

    def on_delete():
        login = login_entry.get()

        if not login:
            messagebox.showinfo("Ошибка", "Нужно ввести логин
сотрудника")
            return

        try:
            cursor = self.connection.cursor()
            message = ''
            res = cursor.callproc('delete_staff', [login, message])
            message = res[1]
            cursor.close()
            self.connection.commit()
            delete_window.destroy()
            messagebox.showinfo("Успех", message)
        except Error as e:
            messagebox.showerror("Ошибка", f"Ошибка при удалении
сотрудника: {e}")

        tk.Button(delete_window, text="OK",
command=on_delete).grid(row=1, columnspan=2)

    def logout(self):
        self.master.destroy()
        root = tk.Tk()
        default_font = tkfont.Font(family="Helvetica", size=font_size,
weight="normal")
        root.option_add("*Font", default_font)
        LoginWindow(root)
        root.mainloop()

class CinemaManagerWindow:
    def __init__(self, master, connection):

```

```

self.master = master
self.connection = connection
self.master.title("Менеджер кинотеатра")

self.view_films_button = tk.Button(master, text="Фильмотека",
command=self.view_films)
self.view_films_button.grid(row=0, column=0)

self.add_schedule_button = tk.Button(master, text="Добавить
фильм в расписание на показ", command=self.add_schedule)
self.add_schedule_button.grid(row=0, column=1)

self.delete_schedule_button = tk.Button(master, text="Удалить
фильм из расписания", command=self.delete_schedule)
self.delete_schedule_button.grid(row=0, column=2)

self.see_schedule_button = tk.Button(master, text="Расписание
показов", command=self.see_schedule)
self.see_schedule_button.grid(row=0, column=3)

self.logout_button = tk.Button(master, text="Выйти",
command=self.logout)
self.logout_button.grid(row=0, column=4)

def view_films(self):
view_window = tk.Toplevel(self.master)
view_window.title("Фильмотека")

tree = ttk.Treeview(view_window, columns=("ID", "Title",
"Genre", "Duration"), show='headings')
tree.heading("ID", text="Номер фильма")
tree.heading("Title", text="Название")
tree.heading("Genre", text="Жанр")
tree.heading("Duration", text="Длительность")
tree.grid(row=0, column=0, columnspan=2)

try:
cursor = self.connection.cursor()
cursor.execute("SELECT * FROM films")
rows = cursor.fetchall()
for row in rows:
tree.insert("", tk.END, values=row)
cursor.close()
except Error as e:
messagebox.showerror("Ошибка", f"Ошибка при получении
данных: {e}")

def add_schedule(self):
add_window = tk.Toplevel(self.master)
add_window.title("Добавить фильм в расписание")

tk.Label(add_window, text="Фильм:").grid(row=0, column=0)
film_combobox = ttk.Combobox(add_window)

```



```

film_combobox.grid(row=0, column=1)

try:
    cursor = self.connection.cursor()
    cursor.execute("SELECT id, title FROM films")
    films = cursor.fetchall()
    film_combobox['values'] = [f"{film[0]}: {film[1]}" for film
in films]
    cursor.close()
except Error as e:
    messagebox.showerror("Ошибка", f"Ошибка при получении
данных: {e}")
    return

    tk.Label(add_window, text="Дата и время показа (YYYY-MM-DD
HH:MM:SS):").grid(row=1, column=0)
    showtime_entry = tk.Entry(add_window)
    showtime_entry.grid(row=1, column=1)

def on_add():
    selected_film = film_combobox.get()
    if not selected_film:
        messagebox.showinfo("Ошибка", "Нужно выбрать фильм")
        return
    film_id = selected_film.split(":")[0]
    showtime = showtime_entry.get()

    if not showtime:
        messagebox.showinfo("Ошибка", "Нужно указать дату и
время показа")
        return

    try:
        cursor = self.connection.cursor()
        message = ''
        res = cursor.callproc('add_schedule', [film_id,
showtime, message])
        message = res[2]
        cursor.close()
        self.connection.commit()
        add_window.destroy()
        messagebox.showinfo("Успех", message)
    except Error as e:
        messagebox.showerror("Ошибка", f"Ошибка при добавлении
фильма в расписание: {e}")

    tk.Button(add_window, text="OK", command=on_add).grid(row=2,
columnspan=2)

def delete_schedule(self):
    delete_window = tk.Toplevel(self.master)
    delete_window.title("Удалить фильм из расписания")

```

```

tk.Label(delete_window, text="Фильм:").grid(row=0, column=0)
film_combobox = ttk.Combobox(delete_window)
film_combobox.grid(row=0, column=1)

try:
    cursor = self.connection.cursor()
    cursor.execute("SELECT id, title FROM films")
    films = cursor.fetchall()
    film_combobox['values'] = [f"{film[0]}: {film[1]}" for film
in films]
    cursor.close()
except Error as e:
    messagebox.showerror("Ошибка", f"Ошибка при получении
данных: {e}")
    return

def on_delete():
    schedule_id = film_combobox.get().split(":")[0]

    if not schedule_id:
        messagebox.showinfo("Ошибка", "Нужно выбрать фильм")
        return

    try:
        cursor = self.connection.cursor()
        message = ''
        res = cursor.callproc('delete_schedule', [schedule_id,
message])
        message = res[1]
        self.connection.commit()
        cursor.close()
        delete_window.destroy()
        messagebox.showinfo("Успех", message)
    except Error as e:
        messagebox.showerror("Ошибка", f"Ошибка при удалении
фильма: {e}")

tk.Button(delete_window, text="OK",
command=on_delete).grid(row=1, columnspan=2)

def see_schedule(self):
    schedule_window = tk.Toplevel(self.master)
    schedule_window.title("Расписание")

    tree = ttk.Treeview(schedule_window, columns=("ID", "Cinema
Name", "Film Title", "Showtime"), show='headings')
    tree.heading("ID", text="Номер показа")
    tree.heading("Cinema Name", text="Название кинотеатра")
    tree.heading("Film Title", text="Название фильма")
    tree.heading("Showtime", text="Дата и время показа")
    tree.grid(row=0, column=0, columnspan=2)

    try:

```

```

        cursor = self.connection.cursor()
        cursor.callproc('see_schedule')
        for result in cursor.stored_results():
            rows = result.fetchall()
            for row in rows:
                tree.insert("", tk.END, values=row)
        cursor.close()
    except Error as e:
        messagebox.showerror("Ошибка", f"Ошибка при получении
данных: {e}")

    def logout(self):
        self.master.destroy()
        root = tk.Tk()
        default_font = tkfont.Font(family="Helvetica", size=font_size,
weight="normal")
        root.option_add("*Font", default_font)
        self.connection.close()
        LoginWindow(root)
        root.mainloop()

class CashierWindow:
    def __init__(self, master, connection):
        self.master = master
        self.connection = connection
        self.master.title("Кассир")

        self.create_widgets()
        self.load_schedules()

    def create_widgets(self):
        self.schedule_label = tk.Label(self.master, text="Выберите
показ:")
        self.schedule_label.grid(row=0, column=0, padx=10, pady=10)

        self.schedule_combobox = ttk.Combobox(self.master)
        self.schedule_combobox.grid(row=0, column=1, padx=10, pady=10)

        self.view_tickets_button = tk.Button(self.master, text="Билеты
на выбранный показ", command=self.view_tickets)
        self.view_tickets_button.grid(row=0, column=2, padx=10,
pady=10)

        self.ticket_entry_label = tk.Label(self.master, text="Введите
номер билета:")
        self.ticket_entry_label.grid(row=1, column=0, padx=10, pady=10)

        self.ticket_number_entry = tk.Entry(self.master)
        self.ticket_number_entry.grid(row=1, column=1, padx=10,
pady=10)

        self.buttons_frame = tk.Frame(self.master)

```

```

        self.buttons_frame.grid(row=2, column=1, padx=10, pady=10)

        self.buy_button = tk.Button(self.buttons_frame, text="Продать
билет", command=self.buy_ticket)
        self.buy_button.grid(row=0, column=0, padx=5, pady=5)

        self.cancel_button = tk.Button(self.buttons_frame,
text="Отменить продажу билета", command=self.cancel_ticket)
        self.cancel_button.grid(row=0, column=1, padx=5, pady=5)

        self.reserve_button = tk.Button(self.buttons_frame,
text="Забронировать билет", command=self.reserve_ticket)
        self.reserve_button.grid(row=1, column=0, padx=5, pady=5)

        self.confirm_reservation_button = tk.Button(self.buttons_frame,
text="Подтвердить бронь билета", command=self.confirm_reservation)
        self.confirm_reservation_button.grid(row=1, column=1, padx=5,
pady=5)

        self.logout_button = tk.Button(self.master, text="Выйти",
command=self.logout)
        self.logout_button.grid(row=3, column=0, columnspan=2, padx=10,
pady=10)

    def reconnect(self):
        try:
            self.connection.ping(reconnect=True, attempts=3, delay=5)
            cursor = self.connection.cursor()
            cursor.execute(f"USE {db_params['database']}")
            cursor.close()
        except Error as e:
            messagebox.showerror("Ошибка", f"Ошибка при подключении к
серверу: {e}")

    def load_schedules(self):
        try:
            if not self.connection.is_connected():
                self.reconnect()
            cursor = self.connection.cursor()
            cursor.execute("CALL see_schedule()")
            schedules = cursor.fetchall()
            self.schedule_combobox['values'] = [f"{s[0]} - {s[1]} -
{s[2]}" for s in schedules]
        except Error as e:
            messagebox.showerror("Ошибка", f"Ошибка при загрузке
доступных показов: {e}")
        finally:
            cursor.close()

    def view_tickets(self):
        ticket_window = tk.Tk()
        schedule_id = int(self.schedule_combobox.get().split(' ')[0])

```

```

        self.ticket_window = TicketsWindow(ticket_window,
self.connection, schedule_id)

    def buy_ticket(self):
        self.perform_ticket_action('sell_ticket', "Продать билет")

    def cancel_ticket(self):
        self.perform_ticket_action('cancel_ticket', "Отменить продажу
билета")

    def reserve_ticket(self):
        phone_number = self.get_phone_number("Reserve Ticket")
        if phone_number:
            self.perform_ticket_action('reserve_ticket', "Забронировать
билет", phone_number)

    def confirm_reservation(self):
        phone_number = self.get_phone_number("Confirm Reservation")
        if phone_number:
            self.perform_ticket_action('confirm_reserve', "Подтвердить
бронь билета", phone_number)

    def get_phone_number(self, action):
        return simpdialog.askstring(action, "Введите номер
телефона:", parent=self.master)

    def perform_ticket_action(self, procedure_name, action_name,
phone_number=None):
        if not self.connection.is_connected():
            self.reconnect()

        schedule_id = int(self.schedule_combobox.get().split(' ')[0])
        ticket_number = self.ticket_number_entry.get()
        if not ticket_number:
            messagebox.showinfo("Ошибка", "Нужно ввести номер билета")
            return

        try:
            cursor = self.connection.cursor()
            message = ""
            if phone_number:
                res = cursor.callproc(procedure_name, [schedule_id,
int(ticket_number), phone_number, message])
                message = res[3]
            else:
                res = cursor.callproc(procedure_name, [schedule_id,
int(ticket_number), message])
                message = res[2]
            messagebox.showinfo("Сообщение", message)
            self.connection.commit()
        except Error as e:
            messagebox.showerror("Ошибка", f"Ошибка при выполнении
действия {action_name}: {e}")

```

```

        finally:
            cursor.close()

    def logout(self):
        self.master.destroy()
        root = tk.Tk()
        default_font = tkfont.Font(family="Helvetica", size=font_size,
weight="normal")
        root.option_add("*Font", default_font)
        LoginWindow(root)
        root.mainloop()

class UserWindow:
    def __init__(self, master, connection):
        self.master = master
        self.connection = connection
        self.master.title("Посетитель")

        self.create_widgets()
        self.load_cinemas()

    def create_widgets(self):
        self.cinema_label = tk.Label(self.master, text="Выберите
кинотеатр:")
        self.cinema_label.grid(row=0, column=0, padx=10, pady=10)

        self.cinema_combobox = ttk.Combobox(self.master)
        self.cinema_combobox.grid(row=0, column=1, padx=10, pady=10)
        self.cinema_combobox.bind("<<ComboboxSelected>>",
self.load_shows)

        self.show_label = tk.Label(self.master, text="Выберите сеанс:")
        self.show_label.grid(row=1, column=0, padx=10, pady=10)

        self.show_combobox = ttk.Combobox(self.master)
        self.show_combobox.grid(row=1, column=1, padx=10, pady=10)

        self.view_shows_button = tk.Button(self.master,
text="Расписание выбранного кинотеатра", command=self.view_shows)
        self.view_shows_button.grid(row=2, column=0, columnspan=2,
padx=10, pady=10)

        self.view_tickets_button = tk.Button(self.master, text="Билеты
на выбранный сеанс", command=self.view_tickets)
        self.view_tickets_button.grid(row=3, column=0, columnspan=2,
padx=10, pady=10)

        self.reserve_ticket_button = tk.Button(self.master,
text="Забронировать билет", command=self.reserve_ticket)
        self.reserve_ticket_button.grid(row=4, column=0, columnspan=2,
padx=10, pady=10)

```

```

        self.cancel_reservation_button = tk.Button(self.master,
text="Отменить бронь билета", command=self.cancel_reservation)
        self.cancel_reservation_button.grid(row=5, column=0,
columnspan=2, padx=10, pady=10)

        self.logout_button = tk.Button(self.master, text="Выйти",
command=self.logout)
        self.logout_button.grid(row=6, column=0, columnspan=2, padx=10,
pady=10)

    def reconnect(self):
        try:
            self.connection.ping(reconnect=True, attempts=3, delay=5)
            cursor = self.connection.cursor()
            cursor.execute(f"USE {db_params['database']}")
            cursor.close()
        except Error as e:
            messagebox.showerror("Ошибка", f"Ошибка при подключении к
серверу: {e}")

    def load_cinemas(self):
        if not self.connection.is_connected():
            self.reconnect()

        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT id, name FROM cinemas")
            cinemas = cursor.fetchall()
            self.cinema_combobox['values'] = [f"{cinema[0]}:
{cinema[1]}" for cinema in cinemas]
        except Error as e:
            messagebox.showerror("Ошибка ", f"Ошибка при загрузке
данных о кинотеатрах: {e}")
        finally:
            cursor.close()

    def load_shows(self, event):
        if not self.connection.is_connected():
            self.reconnect()

        cinema_id = self.get_selected_cinema_id()
        if cinema_id is None:
            return

        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT s.id, f.title FROM schedule s JOIN
films f ON s.film_id = f.id WHERE s.cinema_id = %s", (cinema_id,))
            shows = cursor.fetchall()
            self.show_combobox['values'] = [f"{show[0]}: {show[1]}" for
show in shows]
        except Error as e:

```

```

        messagebox.showerror("Ошибка", f"Ошибка при загрузке
сеансов: {e}")
    finally:
        cursor.close()

def view_shows(self):
    view_shows_window = tk.Toplevel(self.master)
    view_shows_window.title("Расписание показов")

    search_label = tk.Label(view_shows_window, text="Поиск:")
    search_label.grid(row=0, column=0, padx=10, pady=10)
    search_entry = tk.Entry(view_shows_window)
    search_entry.grid(row=0, column=1, padx=10, pady=10)

    search_by_var = tk.StringVar()
    search_by_var.set("Поиск по")
    search_title_rb = tk.Radiobutton(view_shows_window,
text="названию", variable=search_by_var, value="title")
    search_title_rb.grid(row=0, column=2, padx=5, pady=10)
    search_genre_rb = tk.Radiobutton(view_shows_window,
text="жанру", variable=search_by_var, value="genre")
    search_genre_rb.grid(row=0, column=3, padx=5, pady=10)
    search_date_rb = tk.Radiobutton(view_shows_window, text="дате и
времени показа", variable=search_by_var, value="date")
    search_date_rb.grid(row=0, column=4, padx=5, pady=10)

    tree = ttk.Treeview(view_shows_window, columns=("Title",
"Genre", "Date", "Duration"), show='headings')
    tree.heading("Title", text="Название")
    tree.heading("Genre", text="Жанр")
    tree.heading("Date", text="Дата и время показа")
    tree.heading("Duration", text="Длительность")
    tree.grid(row=1, column=0, columnspan=5, padx=10, pady=10)

def update_shows(event=None):
    search_text = search_entry.get()
    search_by = search_by_var.get()

    query = """
        SELECT f.title, f.genre, s.showtime, f.duration
        FROM schedule s
        JOIN films f ON s.film_id = f.id
        WHERE s.cinema_id = %s
    """
    if search_by == "title":
        query += " AND f.title LIKE %s"
    elif search_by == "genre":
        query += " AND f.genre LIKE %s"
    elif search_by == "date":
        query += " AND s.showtime LIKE %s"

    if not self.connection.is_connected():
        self.reconnect()

```



```

        try:
            cursor = self.connection.cursor()
            cursor.execute(query, (self.get_selected_cinema_id(),
f"%{search_text}%"))
            shows = cursor.fetchall()
            for row in tree.get_children():
                tree.delete(row)
            for show in shows:
                tree.insert("", tk.END, values=show)
        except Error as e:
            messagebox.showerror("Ошибка", f"Ошибка при загрузке
данных о сеансах: {e}")
        finally:
            cursor.close()

search_entry.bind("<KeyRelease>", update_shows)

def get_selected_cinema_id(self):
    selected_cinema = self.cinema_combobox.get()
    if selected_cinema:
        return int(selected_cinema.split(":")[0])
    return None

def get_selected_schedule_id(self):
    selected_show = self.show_combobox.get()
    if selected_show:
        return int(selected_show.split(":")[0])
    return None

def view_tickets(self):
    schedule_id = self.get_selected_schedule_id()
    if schedule_id:
        tickets_window = tk.Toplevel(self.master)
        TicketsWindow(tickets_window, self.connection, schedule_id)

def reserve_ticket(self):
    reserve_window = tk.Toplevel(self.master)
    reserve_window.title("Забронировать билет")

    tk.Label(reserve_window, text="Номер билета:").grid(row=0,
column=0, padx=10, pady=10)
    ticket_number_entry = tk.Entry(reserve_window)
    ticket_number_entry.grid(row=0, column=1, padx=10, pady=10)

    tk.Label(reserve_window, text="Номер телефона:").grid(row=1,
column=0, padx=10, pady=10)
    phone_number_entry = tk.Entry(reserve_window)
    phone_number_entry.grid(row=1, column=1, padx=10, pady=10)

    def on_reserve():
        ticket_number = ticket_number_entry.get()
        phone_number = phone_number_entry.get()

```

```

        schedule_id = self.get_selected_schedule_id()

        if not ticket_number or not phone_number:
            messagebox.showinfo("Ошибка", "Все поля должны быть  
заполнены")
            return

        if not self.connection.is_connected():
            self.reconnect()

        try:
            cursor = self.connection.cursor()
            message = ""
            res = cursor.callproc('reserve_ticket', [schedule_id,
int(ticket_number), phone_number, message])
            message = res[3]
            messagebox.showinfo("Сообщение", message)
            self.connection.commit()
        except Error as e:
            messagebox.showerror("Ошибка", f"Ошибка при  
бронировании билета: {e}")
        finally:
            cursor.close()
            reserve_window.destroy()

            tk.Button(reserve_window, text="OK",
command=on_reserve).grid(row=2, columnspan=2, padx=10, pady=10)

    def cancel_reservation(self):
        cancel_window = tk.Toplevel(self.master)
        cancel_window.title("Отменить бронь билета")

        tk.Label(cancel_window, text="Номер билета:").grid(row=0,
column=0, padx=10, pady=10)
        ticket_number_entry = tk.Entry(cancel_window)
        ticket_number_entry.grid(row=0, column=1, padx=10, pady=10)

        tk.Label(cancel_window, text="Номер телефона:").grid(row=1,
column=0, padx=10, pady=10)
        phone_number_entry = tk.Entry(cancel_window)
        phone_number_entry.grid(row=1, column=1, padx=10, pady=10)

    def on_cancel():
        ticket_number = ticket_number_entry.get()
        phone_number = phone_number_entry.get()
        schedule_id = self.get_selected_schedule_id()

        if not ticket_number or not phone_number:
            messagebox.showinfo("Ошибка", "Все поля должны быть  
заполнены")
            return

        if not self.connection.is_connected():

```

```

        self.reconnect()

    try:
        cursor = self.connection.cursor()
        message = ""
        res = cursor.callproc('cancel_reservation',
[schedule_id, int(ticket_number), phone_number, message])
        message = res[3]
        messagebox.showinfo("Сообщение", message)
        self.connection.commit()
    except Error as e:
        messagebox.showerror("Ошибка", f"Ошибка при отмене
брони билета: {e}")
    finally:
        cursor.close()
        cancel_window.destroy()

        tk.Button(cancel_window, text="OK",
command=on_cancel).grid(row=2, columnspan=2, padx=10, pady=10)

    def logout(self):
        self.master.destroy()
        root = tk.Tk()
        default_font = tkfont.Font(family="Helvetica", size=font_size,
weight="normal")
        root.option_add("*Font", default_font)
        LoginWindow(root)
        root.mainloop()

class TicketsWindow:
    def __init__(self, master, connection, schedule_id):
        self.master = master
        self.connection = connection
        self.schedule_id = schedule_id
        self.master.title("Билеты на сеанс")

        self.reconnect()
        total_tickets = self.get_total_tickets()
        self.canvas_width, self.canvas_height =
self.calculate_canvas_size(total_tickets)

        self.canvas = tk.Canvas(self.master, width=self.canvas_width,
height=self.canvas_height, bg='white')
        self.canvas.grid(row=0, column=0, padx=10, pady=10)

        self.draw_tickets()

    def reconnect(self):
        try:
            self.connection.ping(reconnect=True, attempts=3, delay=5)
            cursor = self.connection.cursor()
            cursor.execute(f"USE {db_params['database']}")

```

```

        cursor.close()
    except Error as e:
        messagebox.showerror("Ошибка", f"Ошибка при подключении к серверу: {e}")

    def get_total_tickets(self):
        if not self.connection.is_connected():
            self.reconnect()

        try:
            cursor = self.connection.cursor()

            cursor.execute(f"SELECT count_free_seats({self.schedule_id});")
            free_tickets = int(cursor.fetchone()[0])
            cursor.execute(f"SELECT count_occupied_tickets({self.schedule_id});")
            occupied_tickets = int(cursor.fetchone()[0])

            return free_tickets + occupied_tickets
        except Error as e:
            messagebox.showerror("Ошибка", f"Ошибка при получении данных о билетах: {e}")
            return 0
        finally:
            cursor.close()

    def calculate_canvas_size(self, total_tickets):
        seat_size = 30
        seat_padding = 17
        seats_per_row = 20

        rows = (total_tickets + seats_per_row - 1) // seats_per_row
        width = (seats_per_row * (seat_size + seat_padding)) + seat_padding
        height = (rows * (seat_size + seat_padding)) + seat_padding

        return width, height

    def draw_tickets(self):
        if not self.connection.is_connected():
            self.reconnect()

        try:
            cursor = self.connection.cursor()

            cursor.execute(f"SELECT count_free_seats({self.schedule_id});")
            free_tickets = int(cursor.fetchone()[0])
            cursor.execute(f"SELECT count_occupied_tickets({self.schedule_id});")
            occupied_tickets = int(cursor.fetchone()[0])

```

```

        total_tickets = free_tickets + occupied_tickets
        seat_size = 30
        seat_padding = 17
        seats_per_row = 20

        for ticket_number in range(1, total_tickets + 1):
            x = seat_padding + (ticket_number - 1) % seats_per_row
            * (seat_size + seat_padding)
            y = seat_padding + (ticket_number - 1) // seats_per_row
            * (seat_size + seat_padding)

            cursor.execute(f"SELECT
get_ticket_status({self.schedule_id}, {ticket_number});")
            status = cursor.fetchone()[0]
            color = 'green' if status == 'free' else 'red' if
status == 'purchased' else 'yellow'
            self.canvas.create_rectangle(x, y, x + seat_size, y +
seat_size, fill=color, outline='black')
            self.canvas.create_text(x + seat_size / 2, y +
seat_size / 2, text=str(ticket_number), fill='black')
        except Error as e:
            messagebox.showerror("Ошибка", f"Ошибка при получении
данных о билетах: {e}")
        finally:
            cursor.close()

if __name__ == "__main__":
    root = tk.Tk()
    default_font = tkfont.Font(family="Helvetica", size=font_size,
weight="normal")
    root.option_add("*Font", default_font)
    app = LoginWindow(root)
    root.mainloop()

```