МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Базы данных»
Тема: Нагрузочное тестирование БД.

Студент гр. 3384 Копасова К. А.
Преподаватель Михайлова С. А.

Санкт-Петербург
2025

Цель работы: провести нагрузочное тестирование базы данных для оценки её производительности.

Задание:

Пусть требуется создать программную систему, предназначенную для работников справочной службы кинотеатров города. Такая система должна обеспечивать хранение сведений о кинотеатрах города, о фильмах, которые в них демонстрируются, о сеансах и билетах на эти сеансы. Сведения о кинотеатре — это его название, район города, где расположен кинотеатр, категория, вместимость. Сведения о фильме — это название фильма, режиссёр, оператор, актёры, сыгравшие главные роли, жанр; производство, наличие призов кинофестивалей, продолжительность сеанса, кадр из фильма для рекламы. Кроме того, должна храниться информация о репертуаре кинотеатров на месяц, то есть о том какие фильмы, когда и где демонстрируются, о ценах на билеты и о количестве свободных мест на тот или иной сеанс. На разных сеансах в одном кинотеатре могут идти разные фильмы, а если в кинотеатре несколько залов, то и на одном. Кинотеатр может ввести новый фильм в репертуар или убрать фильм из репертуара. Работник справочной службы может корректировать перечень фильмов, находящихся в прокате – добавлять новые фильмы и снимать с проката, а также перечень кинотеатров, поскольку кинотеатры могут открываться или закрываться, причём иногда временно, например, на ремонт. Цена билета определяется прокатной стоимостью копии фильма, сеансом и категорией кинотеатра. Справочной службе могут потребоваться следующие сведения о текущем состоянии проката фильмов в городе:

1. Репертуар кинотеатра?

2. Адрес и район кинотеатра ?

3. Число свободных мест на данный сеанс в указанном кинотеатре?

4. Цена билетов на данный сеанс в указанном кинотеатре?

5. Жанр, производство и режиссёр данного фильма ?

6. Какие фильмы имеют награды, когда и в каких кинотеатрах они демонстрируются?

7. В каких кинотеатрах в указанный день на указанных сеансах демонстрируется комедия?

Задачи:

Написать скрипт, заполняющий БД большим количеством тестовых данных, рекомендуется использовать faker.js.

Измерить время выполнения запросов, написанных в ЛР3.

a) Проверить для числа записей:
   i. 100 записей в каждой табличке
   ii. 1000 записей
   iii. 10000 записей
   iv. 1000000 записей
   v. можно больше.
b) Все запросы выполнять с фиксированным ограничением на вывод (LIMIT), т.к. запросы без LIMIT всегда будет выполняться O(n) от кол-ва записей
c) Проверить влияние сортировки на скорость выполнения запросов.
d) Для измерения использовать фактическое (не процессорное и т.п.) время. Для node.js есть console.time и console.timeEnd.

Добавить в БД индексы (хотя бы 5 штук). Измерить влияние (или его отсутствие) индексов на скорость выполнения запросов. Обратите внимание на:

a. Скорость сортировки больших табличек

b. Скорость JOIN (но остальные запросы тоже проверьте)

Выполнение работы

1. Установим библиотеку flaker на Python, которая поможет генерировать случайные данные для таблиц:

```
python3.11 -m pip install faker --user
```

2. Для заполнения базы данных были реализованы функции генерации данных для всех таблиц: Cinema, CinemaHalls, Films, Festival, Sessions, Tickets и Prizes. С помощью библиотеки Flaker и random генерируются случайные данные для всех атрибутов в соответствии с их типами. Для таблиц Sessions и Tickets объекты сразу добавляются в сессию SQLAlchemy после создания, что позволяет избежать предупреждений SAWarning. Остальные таблицы добавляются в сессию централизованно в общем методе генерации данных.

Полный файл generate_data.py будет представлен в Приложении 1.

Для проверки генерации выведем 10 записей каждой таблицы:

таблица: cinema

cinema_id	title	city_region	address_cinema	category	total_capacity
1	Реклама	Порог район	д. Тура, алл. Шишкина, д. 802 стр. 41, 785088	обычный	242
2	Господь	Заработать район	к. Сладково, пер. Орловский, д. 47 стр. 9, 722455	обычный	175
3	Выраженный	Ставить район	п. Пенза, ул. Терешковой, д. 20, 782752	обычный	665
4	Иной	Еврейский район	ст. Челябинск, пер. Мельничный, д. 7/9 к. 6/6, 677121	семейный	531
5	Правление	Неправда район	с. Валаам, бул. Авиационный, д. 2/1, 777700	IMAX	266
6	Кожа	Изменение район	г. Ангарск, наб. Казачья, д. 46 к. 3/8, 162942	IMAX	420
7	Медицина	Бабочка район	к. Новый Оскол, пр. Строительный, д. 201 к. 12, 105398	семейный	760
8	Левый	Ставить район	д. Бомнак, бул. Полярный, д. 9 стр. 448, 094551	IMAX	767
9	Сынок	Товар район	ст. Азов (Рост.), алл. Шоссейная, д. 3 стр. 8, 481507	премиум	422
10	Счастье	Тесно район	г. Королев, алл. Красная, д. 41 к. 4, 115183	обычный	859

Рисунок 1 - Таблица cinema.

таблица: films

film_id	title	director	operator	main_actors	genre	production	session_duration	shot_advertising
1	Интернет Мелочь Очередной	Афиноген Ерофеевич Романов	Леонтий Теймуразович Емельянов	Акулина Кузьминична Гурьева, Агап Трофимович Буров, Нонна Харитоновна Степанова	Мелодрама	США	142	единый.jpg
2	Стакан Скользить	Ким Демьянович Селиверстов	Блинова Ангелина Юрьевна	г-жа Аксенова Татьяна Никифоровна, Эрнст Демьянович Журавлев, Тарасова Тамара Ивановна	Биография	Намибия	105	мимо.jpg
3	Палец Приятель	Архипова Милица Евгеньевна	Анна Захаровна Гришина	Василиса Александровна Никитина, Костина Наталья Евгеньевна, Вера Кузьминична Кулакова	Триллер	Мали	181	домашний.jpg
4	Командование Наступать	Веселова Милица Вениаминовна	Евгения Тимофеевна Петрова	Владлен Венедиктович Панов, Романова Клавдия Никифоровна, Никитин Прокофий Богданович	Комедия	Иордания	173	лиловый.jpg
5	Холодно Танцевать Сопровождаться	Ермаков Наркис Игоревич	Лихачев Архип Фадеевич	Чеслав Авдеевич Назаров, Давыдова Глафира Робертовна, Натан Фролович Родионов	Мелодрама	Иордания	74	задрать.jpg
6	Прелесть Палка Исследование	Титов Адам Витальевич	Белоусов Капитон Валерьянович	Зайцев Фадей Владиславович, Федосеева Татьяна Святославовна, Людмила Наумовна Никитина	Боевик	Алжир	118	советовать.jpg
7	Роса Засунуть	Эмилия Ивановна Якушева	Горшкова Евгения Валериевна	г-н Чернов Остромир Ерофеевич, Виктория Руслановна Терентьева, Никон Даниилович Рожков	Боевик	Кипр	157	беспомощный.jpg
8	Добиться Коллектив Волк	Наталья Васильевна Казакова	Колобова Марфа Филипповна	Мечислав Георгиевич Гуцин, Добромысл Илларионович Абрамов, Лукия Юрьевна Игнатьева	Комедия	Австрия	183	багровый.jpg
9	Материя Валюта Единый	Гурий Федосьевич Потапов	Шашкова Раиса Борисовна	Всеволод Владиленович Маслов, Артемий Адамович Смирнов, Филатов Эраст Анатольевич	Фантастика	Эсватини	102	возможно.jpg
10	Успокоиться	Мир Демьянович Котов	Надежда Юрьевна Баранова	Октябрина Альбертовна Лазарева, Фролова Раиса Богдановна, Полина Романовна Кузьмина	Комедия	Оман	193	помолчать.jpg

Рисунок 2 - Таблица films.

таблица: festival

festival_id	title
1	Помимо фестиваль
2	Изучить фестиваль
3	Факультет фестиваль
4	Войти фестиваль
5	Растеряться фестиваль
6	Порог фестиваль
7	Написать фестиваль
8	Выбирать фестиваль
9	Сутки фестиваль
10	Господь фестиваль

Рисунок 3 - Таблица festival.

таблица: cinema_halls

hall_id	cinema_id	title	capacity
1	1	Зал 26	55
2	2	Зал 20	69
3	3	Зал 1	177
4	4	Зал 17	170
5	5	Зал 14	181
6	6	Зал 22	101
7	7	Зал 14	171
8	8	Зал 27	173
9	9	Зал 26	191
10	10	Зал 15	137

Рисунок 4 - Таблица cinema_halls.

таблица: sessions

session_id	hall_id	film_id	date_session	start_time	end_time
1	4	61	2025-10-28	22:13:08	23:52:08
2	74	65	2025-11-03	15:04:07	16:40:07
3	3	20	2025-11-26	05:44:26	08:28:26
4	11	17	2025-12-07	13:58:26	15:30:26
5	95	90	2025-11-15	03:42:19	05:17:19
6	44	3	2025-11-11	13:59:53	17:00:53
7	40	8	2025-11-23	13:57:40	17:00:40
8	77	89	2025-10-22	18:33:26	19:45:26
9	87	47	2025-12-05	14:50:44	16:29:44
10	13	40	2025-12-14	01:24:05	03:32:05

Рисунок 5 - Таблица sessions.

таблица: tickets

ticket_id	session_id	row_number	place_number	sold_out	cost
1	1	6	11	1	600
2	1	4	17	1	600
3	1	3	4	0	500
4	1	2	19	0	600
5	1	1	6	1	300
6	1	6	25	0	600
7	1	3	18	1	300
8	1	2	3	1	500
9	1	6	8	0	300
10	1	1	8	1	500

Рисунок 6 - Таблица tickets.

таблица: prizes

prizes_id	festival_id	film_id	title
1	67	2	Разнообразный приз
2	91	8	Мгновение приз
3	71	18	Желание приз
4	48	22	Самостоятельно приз
5	10	26	Мелочь приз
6	92	38	При приз
7	82	46	Уничтожение приз
8	36	53	Академик приз
9	41	57	Хотеть приз
10	42	59	Покинуть приз

Рисунок 7 - Таблица prizes.

3. Измерим время выполнения всех запросов, написанных в ЛР 3, для разного числа записей в таблицах:

Запрос 1: Репертуар кинотеатра?

Запрос 2: Адрес и район кинотеатра?

Запрос 3: Число свободных мест на данный сеанс в указанном кинотеатре?

Запрос 4: Цена билетов на данный сеанс в указанном кинотеатре?

Запрос 5: Жанр, производство и режиссер данного фильма?

Запрос 6: Какие фильмы имеют награды, когда и в каких кинотеатрах они демонстрируются?

Запрос 7: В каких кинотеатрах в указанный день на указанных сеансах демонстрируется комедия?

В таб. 1 результат измерений без сортировки, а в таб. 2 - с сортировкой.

Таблица 1 - Время выполнения запросов для разного числа записей без сортировки.

Число записей	Запрос 1	Запрос 2	Запрос 3	Запрос 4	Запрос 5	Запрос 6	Запрос 7
100	0.001050/ 0.001334	0.000549/ 0.000258	0.001049/ 0.000386	0.000944/ 0.000473	0.000609/ 0.000241	0.011173/ 0.000639	0.000921/ 0.000684
1 000	0.002572/ 0.001499	0.000839/ 0.000436	0.001448/ 0.000410	0.000927/ 0.000566	0.000663/ 0.000275	0.107756/ 0.000658	0.000995/ 0.000741
10 000	0.004973/ 0.001733	0.000652/ 0.000323	0.001694/ 0.000421	0.000808/ 0.000661	0.000682/ 0.000238	1.038983/ 0.000637	0.001085/ 0.000877
100 000	0.037674/ 0.001413	0.000661/ 0.000343	0.001002/ 0.000477	0.000891/ 0.000507	0.000668/ 0.000239	11.008285/ 0.000701	0.001227/ 0.000918
1 000 000	0.415381/ 0.001938	0.000829/ 0.000438	0.002050/ 0.000972	0.001427/ 0.000593	0.000857/ 0.000320	114.070016 /0.000646	0.001827/ 0.000914

Таблица 2 - Время выполнения запросов для разного числа записей с сортировкой.

Число записей	Запрос 1	Запрос 2	Запрос 3	Запрос 4	Запрос 5	Запрос 6	Запрос 7
100	0.001119/ 0.001641	0.000593/ 0.000909	0.001044/ 0.000386	0.000852/ 0.000678	0.000580/ 0.000241	0.011305/ 0.000914	0.001044/ 0.000936

1 000	0.002505/ 0.001812	0.000758/ 0.000958	0.001374/ 0.000410	0.000885/ 0.000781	0.000667/ 0.000275	0.108765/ 0.000950	0.001148/ 0.001009
10 000	0.005086/ 0.002106	0.000649/ 0.000649	0.001716/ 0.000421	0.000891/ 0.000883	0.000701/ 0.000238	1.081416/ 0.000879	0.001187/ 0.001121
100 000	0.037960/ 0.002180	0.000672/ 0.000950	0.000919/ 0.000477	0.000950/ 0.000731	0.000648/ 0.000239	11.707313/ 0.000961	0.001191/ 0.001276
1 000 000	0.362786/ 0.002250	0.000786/ 0.000972	0.001021/ 0.000972	0.001019/ 0.000891	0.000762/ 0.000320	115.198978 /0.000885	0.001318/ 0.001283

Время выполнения всех запросов с ростом числа записей незначительно увеличивается. Запросы 2-6 показывают близкие и относительно стабильные значения времени выполнения при росте числа записей, в то время как запросы 1 и 7 достаточно ресурсоемкие из-за сложных условий выборки.

Сортировка добавляет стабильную небольшую нагрузку для всех запросов.

Добавим ко всем таблицам по одному индексу (пример добавления индекса в models.py представлен на рис. 8).

```python
class Cinema(Base):
    __tablename__ = 'cinema'

    cinema_id = Column(Integer, primary_key=True, autoincrement=True)
    title = Column(String(100), nullable=False)
    city_region = Column(String(100), nullable=False)
    address_cinema = Column(String(100), nullable=False)
    category = Column(String(30))
    total_capacity = Column(Integer, nullable=False)

    __table_args__ = (CheckConstraint('total_capacity > 0'), Index('idx_cinema_title', 'title'),)

    halls = relationship("CinemaHalls", back_populates="cinema", cascade="all, delete")
```

Рисунок 8 - Пример добавления индекса в таблицу Cinema.

Измерим время выполнения запросов для разного числа записей без сортировки (таб. 3) и с сортировкой (таб. 4).

Таблица 3 - Время выполнения запросов для разного числа записей с использованием индексов без сортировки.

Число записей	Запрос 1	Запрос 2	Запрос 3	Запрос 4	Запрос 5	Запрос 6	Запрос 7
100	0.000907/ 0.001273	0.000637/ 0.000276	0.000987/ 0.000397	0.000934/ 0.000456	0.000614/ 0.000213	0.010906/ 0.000549	0.001235/ 0.000660
1 000	0.001148/ 0.001276	0.000594/ 0.000232	0.000897/ 0.000399	0.000792/ 0.000462	0.000638/ 0.000188	0.094577/ 0.000544	0.001271/ 0.000760
10 000	0.004071/ 0.001372	0.000624/ 0.000306	0.001162/ 0.000390	0.000873/ 0.000447	0.000633/ 0.000282	1.053571/ 0.000545	0.001224/ 0.000794
100 000	0.036871/ 0.002389	0.000676/ 0.000348	0.001038/ 0.000654	0.000992/ 0.000665	0.000660/ 0.000288	10.525647/ 0.000835	0.001275/ 0.000947
1 000 000	0.259974/ 0.001491	0.000626/ 0.000521	0.001378/ 0.000420	0.000978/ 0.000558	0.000663/ 0.000304	108.486316 /0.000555	0.002054/ 0.001129

Таблица 4 - Время выполнения запросов для разного числа записей с использованием индексов с сортировкой.

Число записей	Запрос 1	Запрос 2	Запрос 3	Запрос 4	Запрос 5	Запрос 6	Запрос 7
100	0.000947/ 0.001577	0.000613/ 0.000525	0.000983/ 0.000397	0.000954/ 0.000686	0.000595/ 0.000213	0.012091/ 0.000694	0.001229/ 0.000852
1 000	0.001231/	0.000607/	0.000889/	0.000793/	0.000596/	0.103006/	0.001179/

	0.001511	0.000537	0.000399	0.000692	0.000188	0.000687	0.001001
10 000	0.004283/ 0.001676	0.000636/ 0.000555	0.000936/ 0.000390	0.000875/ 0.000672	0.000617/ 0.000282	1.079993/ 0.000689	0.001339/ 0.001024
100 000	0.041050/ 0.002715	0.000653/ 0.000618	0.000858/ 0.000654	0.000917/ 0.000935	0.000640/ 0.000288	11.047061/ 0.001022	0.001232/ 0.001248
1 000 000	0.220763/ 0.002751	0.000648/ 0.000820	0.000944/ 0.000420	0.001009/ 0.001011	0.000587/ 0.000304	112.703760 /0.001218	0.001492/ 0.001440

Для таблиц с индексами время выполнения большинства запросов становиться более стабильным при росте числа записей. Наиболее стабильными являются запросы 2–5, для которых время выполнения практически не зависит от объёма данных. Наиболее ресурсоёмким становится запрос 1, для которого наблюдаются наибольшие значения времени выполнения и более заметная зависимость от числа записей; запрос 7 также относится к более тяжёлым, но выполняется быстрее, чем запрос 1.

Добавление сортировки приводит к небольшому и стабильному увеличению времени выполнения всех запросов.

По сравнению с таблицами 1 и 2 (без индексов), в таблицах 3 и 4 (с индексами) время выполнения запросов становится более стабильным и менее зависимым от объёма данных.

Вывод

В ходе лабораторной работы проведено нагрузочное тестирование базы данных для оценки ее производительности при различных условиях.

Полученные результаты показывают, что при увеличении объёма данных время выполнения большинства запросов возрастает незначительно, однако для более сложных запросов наблюдается более выраженная зависимость от числа записей. Использование индексов позволяет сделать время выполнения запросов более стабильным и снизить влияние роста объёма данных, а добавление сортировки приводит лишь к умеренному увеличению времени выполнения без существенного ухудшения общей производительности.

Полученные результаты подтверждают важность правильной индексации для обеспечения высокой производительности базы данных и указывают на необходимость дальнейшей оптимизации сложных запросов через создание специализированных составных индексов.

ПРИЛОЖЕНИЕ 1

Ссылка на pull request: https://github.com/moevm/sql-2025-3384/pull/41.

Файл *models.py*:

```
  from sqlalchemy import Column, Integer, String, Date, Time, Text, ForeignKey,
CheckConstraint, Index
  from sqlalchemy.orm import declarative_base, relationship

  Base = declarative_base()

  class Cinema(Base):
      __tablename__ = 'cinema'

      cinema_id = Column(Integer, primary_key=True, autoincrement=True)
      title = Column(String(100), nullable=False)
      city_region = Column(String(100), nullable=False)
      address_cinema = Column(String(100), nullable=False)
      category = Column(String(30))
      total_capacity = Column(Integer, nullable=False)

                __table_args__    =    (CheckConstraint('total_capacity    >    0'),
Index('idx_cinema_title', 'title'),)

      halls = relationship("CinemaHalls", back_populates="cinema", cascade="all,
delete")

  class Films(Base):
      __tablename__ = 'films'

      film_id = Column(Integer, primary_key=True, autoincrement=True)
      title = Column(String(100), nullable=False)
      director = Column(String(100), nullable=False)
      operator = Column(String(100))
      main_actors = Column(Text)
      genre = Column(String(30))
      production = Column(String(100))
      session_duration = Column(Integer, nullable=False)
      shot_advertising = Column(String(255))

                __table_args__    =    (CheckConstraint('session_duration    >    0'),
Index('idx_films_title', 'title'),)

       sessions = relationship("Sessions", back_populates="film", cascade="all,
delete")
         prizes  =  relationship("Prizes",  back_populates="film",  cascade="all,
delete")

  class Festival(Base):
      __tablename__ = 'festival'

      festival_id = Column(Integer, primary_key=True, autoincrement=True)
      title = Column(String(100), nullable=False)

      __table_args__ = (Index('idx_festival_title', 'title'),)

       prizes = relationship("Prizes", back_populates="festival", cascade="all,
delete")

  class CinemaHalls(Base):
      __tablename__ = 'cinema_halls'
```

```
      hall_id = Column(Integer, primary_key=True, autoincrement=True)
                cinema_id    =    Column(Integer,    ForeignKey('cinema.cinema_id',
ondelete='CASCADE'), nullable=False)
      title = Column(String(100), nullable=False)
      capacity = Column(Integer, nullable=False)

                          __table_args__      =      (CheckConstraint('capacity      >
0'),Index('idx_hall_cinemaid', 'cinema_id'),)

      cinema = relationship("Cinema", back_populates="halls")
       sessions = relationship("Sessions", back_populates="hall", cascade="all,
delete")

  class Sessions(Base):
      __tablename__ = 'sessions'

      session_id = Column(Integer, primary_key=True, autoincrement=True)
              hall_id    =    Column(Integer,    ForeignKey('cinema_halls.hall_id',
ondelete='CASCADE'), nullable=False)
      film_id = Column(Integer, ForeignKey('films.film_id', ondelete='CASCADE'),
nullable=False)
      date_session = Column(Date, nullable=False)
      start_time = Column(Time, nullable=False)
      end_time = Column(Time, nullable=False)

      __table_args__ = (Index('idx_sessions_filmid', 'film_id'),)

      hall = relationship("CinemaHalls", back_populates="sessions")
      film = relationship("Films", back_populates="sessions")
       tickets = relationship("Tickets", back_populates="session", cascade="all,
delete")

  class Tickets(Base):
      __tablename__ = 'tickets'

      ticket_id = Column(Integer, primary_key=True, autoincrement=True)
             session_id    =    Column(Integer,    ForeignKey('sessions.session_id',
ondelete='CASCADE'), nullable=False)
      row_number = Column(Integer, nullable=False)
      place_number = Column(Integer, nullable=False)
      sold_out = Column(String(1), default='0')
      cost = Column(Integer, nullable=False)

      __table_args__ = (
          CheckConstraint('row_number > 0'),
          CheckConstraint('place_number > 0'),
          CheckConstraint('cost >= 0'),
          Index('idx_ticket_sessionid', 'session_id'),
      )

      session = relationship("Sessions", back_populates="tickets")

  class Prizes(Base):
      __tablename__ = 'prizes'

      prizes_id = Column(Integer, primary_key=True, autoincrement=True)
            festival_id    =    Column(Integer,    ForeignKey('festival.festival_id',
ondelete='CASCADE'), nullable=False)
      film_id = Column(Integer, ForeignKey('films.film_id', ondelete='CASCADE'),
nullable=False)
      title = Column(String(100), nullable=False)

      __table_args__ = (Index('idx_prizes_filmid', 'film_id'),)
```

```
    festival = relationship("Festival", back_populates="prizes")
    film = relationship("Films", back_populates="prizes")
```

Файл *generate_data.py:*

```
from faker import Faker
from random import randint, choice
from datetime import timedelta
import math
from db_config import Session
from models import Cinema, Films, Festival, CinemaHalls, Sessions, Tickets,
Prizes

fake = Faker("ru_RU")

def generate_cinemas(n):
    data = []
    for _ in range(n):
        data.append(Cinema(
            title=fake.word().capitalize(),
            city_region=f"{fake.word().capitalize()} район",
            address_cinema=fake.address().replace("\n", ", "),
            category=choice(["премиум", "обычный", "семейный", "IMAX"]),
            total_capacity=randint(150, 900)
        ))

    return data

def generate_halls(cinemas, max_halls_total):
    halls = []
    total_halls_created = 0

    # каждому кинотеатру даем хотя бы по 1 залу
    for cinema in cinemas:
        if total_halls_created >= max_halls_total:
            break
        hall_capacity = randint(50, 200)
        hall_number = randint(1, 30)
        halls.append(CinemaHalls(
            cinema=cinema,
            title=f"Зал {hall_number}",
            capacity=hall_capacity
        ))
        total_halls_created += 1

    # распределяем оставшиеся залы случайно
    remaining_halls = max_halls_total - total_halls_created

    while remaining_halls > 0:
        cinema = choice(cinemas)
        hall_capacity = randint(50, 200)
        hall_number = randint(1, 30)
        halls.append(CinemaHalls(
            cinema=cinema,
            title=f"Зал {hall_number}",
            capacity=hall_capacity
        ))
        total_halls_created += 1
        remaining_halls -= 1

    return halls

def generate_films(n):
```

```
        films = []
        for _ in range(n):
            films.append(Films(
                title=" ".join(fake.words(nb=randint(1, 3))).title(),
                director=fake.name(),
                operator=fake.name(),
                main_actors=", ".join(fake.name() for _ in range(3)),
                genre=choice(["Комедия", "Драма", "Боевик", "Фантастика", "Ужасы",
"Триллер", "Мелодрама", "Биография"]),
                production=fake.country(),
                session_duration=randint(60, 200),
                shot_advertising=fake.file_name(extension="jpg")
            ))

        return films

    def generate_festivals(n):
        festivals = []
        for _ in range(n):
            festivals.append(Festival(
                title=f"{fake.word().capitalize()} фестиваль"
            ))

        return festivals

    def generate_sessions(films, halls, n, session_obj):
        sessions = []
        for _ in range(n):
            film = choice(films)
            hall = choice(halls)

            start = fake.date_time_between(start_date="-30d", end_date="+30d")
            end = start + timedelta(minutes=film.session_duration)

            sess = Sessions(
                hall=hall,
                film=film,
                date_session=start.date(),
                start_time=start.time(),
                end_time=end.time()
            )
            sessions.append(sess)
            session_obj.add(sess)

        session_obj.commit()
        return sessions

    def generate_tickets(sessions, session_obj, max_tickets=None, max_rows=15):
        tickets = []
        total_generated = 0

        for sess in sessions:
            hall_capacity = sess.hall.capacity

            rows = randint(1, max_rows)
            places_per_row = math.ceil(hall_capacity / rows)

            tickets_for_session = min(hall_capacity, max_tickets - total_generated)
if max_tickets else hall_capacity

            for _ in range(tickets_for_session):
                row = randint(1, rows)
                place = randint(1, places_per_row)
```

```
                ticket = Tickets(
                    session=sess,
                    row_number=row,
                    place_number=place,
                    sold_out=choice(["0", "1"]),
                    cost=choice([300, 400, 500, 600])
                )
                tickets.append(ticket)
                session_obj.add(ticket)
                total_generated += 1

            if max_tickets and total_generated >= max_tickets:
                break

        session_obj.commit()
        return tickets

    def generate_prizes(films, festivals, prizes_n):
        prizes = []
        for _ in range(prizes_n):
            prizes.append(Prizes(
                film=choice(films),
                festival=choice(festivals),
                title=fake.word().capitalize() + " приз"
            ))
        return prizes

    def generate_database(cinemas_n=100, max_halls_total = 100, max_tickets=100,
films_n=100, prizes_n=100,festivals_n=100, sessions_n=100):
        session = Session()
        print("генерация кинотеатров...")
        cinemas = generate_cinemas(cinemas_n)
        session.add_all(cinemas)
        session.commit()

        print("генерация фильмов...")
        films = generate_films(films_n)
        session.add_all(films)
        session.commit()

        print("генерация залов...")
        halls = generate_halls(cinemas, max_halls_total)
        session.add_all(halls)
        session.commit()

        print("генерация фестивалей...")
        festivals = generate_festivals(festivals_n)
        session.add_all(festivals)
        session.commit()

        print("генерация сеансов...")
        sessions = generate_sessions(films, halls, sessions_n, session)

        print("генерация билетов...")
        tickets = generate_tickets(sessions, session, max_tickets)

        print("генерация наград...")
        prizes = generate_prizes(films, festivals,prizes_n)
        session.add_all(prizes)
        session.commit()
```

Файл *questions.py:*

```
  from sqlalchemy import func, distinct
  from sqlalchemy.orm import sessionmaker
  from tabulate import tabulate
  from models import *
  from db_config import *
  import time

  def repertoire(cinema_name, sort=False, limit=None):
      start = time.perf_counter()
      query = (session.query(distinct(Films.title), Films.genre, Films.director,
Films.session_duration)
                .join(Sessions, Films.film_id == Sessions.film_id)
                .join(CinemaHalls, Sessions.hall_id == CinemaHalls.hall_id)
                .join(Cinema, CinemaHalls.cinema_id == Cinema.cinema_id)
                .filter(Cinema.title == cinema_name))

      end = time.perf_counter()
      print("Время выполения запроса 1 БЕЗ СОРТИРОВКИ: ", end - start)

      if sort:
          query = query.order_by(Films.title)

      end = time.perf_counter()
      print("Время выполения запроса 1 С СОРТИРОВКОЙ: ", end - start)

      if limit is not None:
          query = query.limit(limit)

      result = query.all()

      headers = ["Название фильма", "Жанр", "Режиссер", "Продолжительность
(мин)"]
      return tabulate(result, headers=headers, tablefmt="fancy_grid")


  def cinema_address(cinema_name, sort=False, limit=None):
      start = time.perf_counter()
      query       =       session.query(Cinema.title,       Cinema.city_region,
Cinema.address_cinema)\
          .filter(Cinema.title == cinema_name)

      end = time.perf_counter()
      print("Время выполения запроса 2 БЕЗ СОРТИРОВКИ: ", end - start)

      if sort:
          query = query.order_by(Cinema.title)

      end = time.perf_counter()
      print("Время выполения запроса 2 С СОРТИРОВКОЙ: ", end - start)

      if limit is not None:
          query = query.limit(limit)

      result = query.first()

      headers = ["Кинотеатр", "Район", "Адрес"]
      return tabulate([result], headers=headers, tablefmt="fancy_grid")


  def free_places(cinema_name, session_id, sort=False, limit=None):
      start = time.perf_counter()
      query = (session.query(func.count(Tickets.ticket_id))
               .join(Sessions)
               .join(CinemaHalls)
```

```
                .join(Cinema)
                .filter(Cinema.title == cinema_name, Sessions.session_id ==
session_id, Tickets.sold_out == '0'))

      end = time.perf_counter()
      print("Время выполения запроса 3 БЕЗ СОРТИРОВКИ: ", end - start)

      if limit is not None:
          query = query.limit(limit)

      count = query.scalar()

      headers = ["Свободные места"]
      return tabulate([[count]], headers=headers, tablefmt="fancy_grid")


  def ticket_prices(cinema_name, session_id, sort=False, limit=None):
      start = time.perf_counter()
      query = (session.query(distinct(Tickets.cost), Cinema.title,
Sessions.start_time)
                .join(Sessions, Tickets.session_id == Sessions.session_id)
                .join(CinemaHalls, Sessions.hall_id == CinemaHalls.hall_id)
                .join(Cinema, CinemaHalls.cinema_id == Cinema.cinema_id)
                .filter(Cinema.title == cinema_name, Sessions.session_id ==
session_id))

      end = time.perf_counter()
      print("Время выполения запроса 4 БЕЗ СОРТИРОВКИ: ", end - start)

      if sort:
          query = query.order_by(Tickets.cost)

      end = time.perf_counter()
      print("Время выполения запроса 4 С СОРТИРОВКОЙ: ", end - start)

      if limit is not None:
          query = query.limit(limit)

      result = query.all()

      headers = ["Цена билета", "Название кинотеатра", "Время начала"]
      return tabulate(result, headers=headers, tablefmt="fancy_grid")


  def film_info(film_title, sort=False, limit=None):
      start = time.perf_counter()
      query = session.query(Films.title, Films.genre, Films.production,
Films.director)\
          .filter(Films.title == film_title)

      end = time.perf_counter()
      print("Время выполения запроса 5 БЕЗ СОРТИРОВКИ: ", end - start)

      if limit is not None:
          query = query.limit(limit)

      result = query.first()

      headers = ["Фильм", "Жанр", "Производство", "Режиссер"]
      return tabulate([result], headers=headers, tablefmt="fancy_grid")


  def films_with_prizes(sort=False, limit=None):
      start = time.perf_counter()
```

```
      query    =    (session.query(Films.title,    Prizes.title.label("Награда"),
Festival.title.label("Фестиваль"),
                               Sessions.date_session, Sessions.start_time,
                               Cinema.title.label("Кинотеатр"),
CinemaHalls.title.label("Зал"),
                               Cinema.address_cinema)
                .join(Prizes, Films.film_id == Prizes.film_id)
                .join(Festival, Prizes.festival_id == Festival.festival_id)
                .join(Sessions, Films.film_id == Sessions.film_id)
                .join(CinemaHalls, Sessions.hall_id == CinemaHalls.hall_id)
                .join(Cinema, CinemaHalls.cinema_id == Cinema.cinema_id))

      end = time.perf_counter()
      print("Время выполения запроса 6 БЕЗ СОРТИРОВКИ: ", end - start)

      if sort:
          query     =     query.order_by(Films.title,     Sessions.date_session,
Sessions.start_time)

      end = time.perf_counter()
      print("Время выполения запроса 6 С СОРТИРОВКОЙ: ", end - start)

      if limit is not None:
          query = query.limit(limit)

      result = query.all()

      headers = ["Фильм", "Награда", "Фестиваль", "Дата сеанса", "Время начала",
"Кинотеатр", "Зал", "Адрес"]
      return tabulate(result, headers=headers, tablefmt="fancy_grid")


  def comedy_on_day(date, session_ids, sort=False, limit=None):
      start = time.perf_counter()
      query           =           (session.query(Cinema.title.label("Кинотеатр"),
Films.title.label("Фильм"), Films.genre,
                               Sessions.session_id,
CinemaHalls.title.label("Зал"),
                               Sessions.date_session,        Sessions.start_time,
Sessions.end_time)
                .join(Films, Sessions.film_id == Films.film_id)
                .join(CinemaHalls, Sessions.hall_id == CinemaHalls.hall_id)
                .join(Cinema, CinemaHalls.cinema_id == Cinema.cinema_id)
                .filter(Films.genre == 'Комедия', Sessions.date_session == date,
                        Sessions.session_id.in_(session_ids)))

      end = time.perf_counter()
      print("Время выполения запроса 7 БЕЗ СОРТИРОВКИ: ", end - start)

      if sort:
          query = query.order_by(Sessions.start_time)

      end = time.perf_counter()
      print("Время выполения запроса 7 С СОРТИРОВКОЙ: ", end - start)

      if limit is not None:
          query = query.limit(limit)

      result = query.all()

      headers = ["Кинотеатр", "Фильм", "Жанр", "Сеанс", "Зал", "Дата", "Начало",
"Окончание"]
      return tabulate(result, headers=headers, tablefmt="fancy_grid")
```

```
  def all_questions():
      # примеры запросов
      print("\n1. Репертуар кинотеатра? (кинотеатр Аврора)")
      print(repertoire("Аврора"))

      print("\n2. Адрес и район кинотеатра? (кинотеатр Аврора)")
      print(cinema_address("Аврора"))

      print("\n3. Число свободных мест на данный сеанс в указанном кинотеатре?
(кинотеатр Аврора, сеанс 1)")
      print(free_places("Аврора", 1))

      print("\n4.  Цена  билетов  на  данный  сеанс  в  указанном  кинотеатре?
(кинотеатр Аврора, сеанс 1)")
      print(ticket_prices("Аврора", 1))

      print("\n5. Жанр, производство и режиссер данного фильма? (Барби)")
      print(film_info("Барби"))

      print("\n6. Какие фильмы имеют награды, когда и в каких кинотеатрах они
демонстрируются?")
      print(films_with_prizes())

      print("\n7.  В каких кинотеатрах в указанный день на указанных сеансах
демонстрируется комедия? (день 2025-10-21, сеансы 1, 2, 3, 6 и 7)")
      print(comedy_on_day("2025-10-21", [1,2,3,6,7]))
```

Файл *main.py*

```
  from create_table import *
  from insert_data import *
  from questions import *
  from show_table import *
  from delete_table import *
  from generate_data import *
  from measure_time import *

  def main():
      # # удаляем таблицы
      delete_all_tables()

      # создаём таблицы
      create_db()

      # генерируем данные
      generate_database()

      # показываем таблицы
      show_all_table(10)

      # замеряем время всех запросов
      measure_all()

  if __name__ == "__main__":
      main()
```