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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Базы данных»

Тема: Нагрузочное тестирование БД.

Студент гр. 3384 _____ Рудаков А.Л.

Преподаватель _____ Михайлова С.В.

Санкт-Петербург

2025

Цель работы.

Протестировать реализованные ранее запросы к БД на разных количествах данных, одновременно находящихся в таблицах..

Задание.

Вариант 22.

Пусть требуется создать программную систему для поиска вакансий (аналог hh.ru). Такая система должна обеспечивать хранение сведений о работодателях и работниках. Эти сведения включают в себя (для работника) - паспортные работника, данные трудовой книжки, ИНН, дата рождения, информацию о среднем/высшем(их) образованиях, дата поступления на работу, в институт, информация об предыдущей работе(ах) из трудовой книжки. Данные трудовой книжки – это ее номер и дата выдачи, а также даты и номера приказов о зачислении и увольнении, о переходе в другое подразделение или об изменении должности. Кроме того, для работник может создать 1/несколько резюме, с указанием желаемой должности, ЗП, свои умения/навыки. Работодатель имеет возможность создавать/удалять/помещать в архив вакансии. Вакания имеет название, ЗП, должность, адрес, требования, условия, комментарий, требуемый опыт. У работодателя есть страница с указанием информации о себе - название, фото, описание, файлы презентации, сфера деятельности (IT, финансовая и т.п.), количество вакансий (формируется на основе списка вакансий).

Система должна давать ответы на следующие вопросы:

- Какие вакансии есть у данной компании?
- Какая вакансия подходит мне по названию?
- Сколько поблизости вакансий от меня (указание улицы)?
- Какие вакансии были помещены в архив у компании?
- Средняя ЗП каждого работодателя?
- Сколько работников ищут работу, имея высшее образование?

- Сколько работников имело более 3-х мест работы?
- Какие вакансии имеют ЗП более 100 000р и не требуют опыта работы?

Задачи:

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

Измерить время выполнения запросов, написанных в ЛР3.

Проверить для числа записей:

- 100 записей в каждой табличке
- 1.000 записей
- 1.0000 записей
- 1.000.000 записей
- можно больше.

Все запросы выполнять с фиксированным ограничением на вывод (LIMIT), т.к. запросы без LIMIT всегда будет выполняться O(n) от кол-ва записей

Проверить влияние сортировки на скорость выполнения запросов.

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:

- Скорость сортировки больших табличек
- Скорость JOIN (но остальные запросы тоже проверьте)

Выполнение работы.

Для заполнения таблиц тестовыми данными были написаны методы generate_* для генерации данных всех классов, требуемых для заполнения таблиц. Для генерации был использован Faker и для генерации некоторых полей (*name* для *education*, *description* для *skill* и *requirement*, *name* для *field_of_activity*, *description* для *condition*, *name* для *post*) добавлены кастомные provider'ы с функциями генерации.

В табл. 1. представлены результаты замеров среднего времени в секундах выполнения запросов на разных количествах данных, находящихся в таблицах.

Таблица 1 – результаты замеров

Размер, Номер запроса	1	2	3	4	5	6	7	8
100	0.0132	0.0122	0.0122	0.0136	0.0132	0.0144	0.0146	0.0134
1 000	0.0140	0.0136	0.0126	0.0138	0.0145	0.0149	0.0149	0.0140
10 000	0.0158	0.0138	0.0160	0.0164	0.0239	0.0208	0.0246	0.0143
100 000	0.0233	0.0138	0.0475	0.0272	0.1952	0.0991	0.1711	0.0152
1 000 000	0.0561	0.0141	0.1267	0.0789	1.5462	0.3267	0.9308	0.0155

В табл. 2 представлены результаты замеров среднего времени в секундах выполнения запросов с количеством элементов в таблицах равным 1000000, с использованием в запросах сортировок.

Таблица 2 – результаты тестирования с применением сортировки

Размер, Номер запроса	1	2	3	4	5	6	7	8
1 000 000	0.0575	0.0849	0.1283	0.0776	1.5861	0.3276	0.9379	0.7625

Из результатов, полученных в табл. 2. можно заметить, что времена выполнения запросов 1, 3, 4, 5, 6, 7 не изменились, что связано с тем, что в запросе 5 и так присутствует сортировка, а в остальные указанные запросы являются достаточно “точечными”, поэтому их полный вывод не такой большой, из-за чего сортировка не сильно влияет на время. Однако у запросов 2 и 8 время кратно увеличилось, так как их полный вывод является большим и его сортировка занимает время.

Добавление индексов в таблицы для каждого запроса представлено на рис. 1.

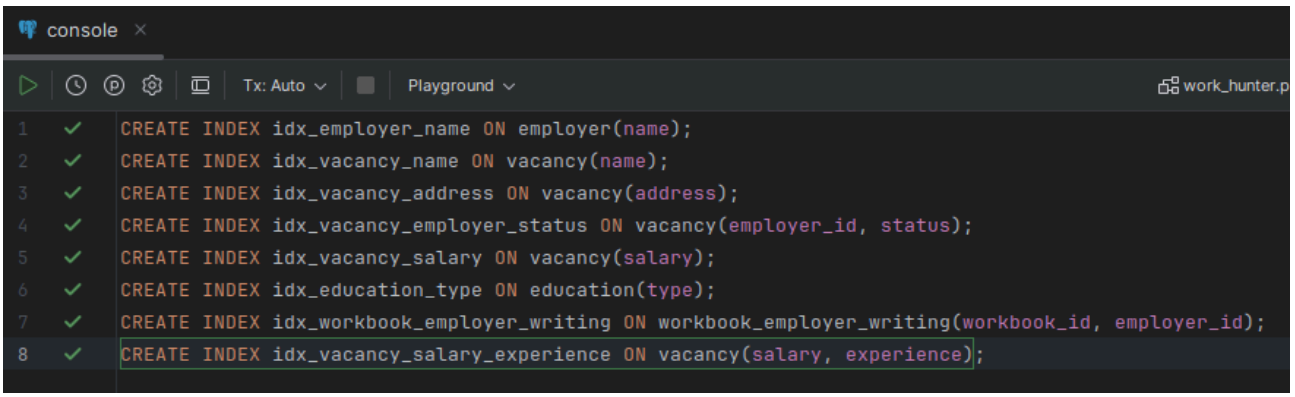

Рисунок 1 - Добавление индексов к таблицам

В табл. 3 представлены результаты замеров среднего времени в секундах выполнения запросов с количеством элементов в таблицах равным 1000000, с использованием в запросах сортировок и индексов.

Таблица 3 – результаты тестирования с применением сортировки и использованием индексов

Размер, Номер запроса	1	2	3	4	5	6	7	8
1 000 000	0.0148	0.0189	0.1246	0.0144	1.5306	0.3201	0.9452	0.0146

Из результатов, полученных в табл. 3. можно заметить, что времена выполнения запросов 3, 5, 6, 7 не изменились, возможно это связано с неправильным выбором индексов или их “ненадобности” в данных запросах. В запросах 1, 2, 4, 8 время заметно уменьшилось, что значит, что индексы подобраны хорошо.

В табл. 4. представлены ускорения, полученные благодаря индексам (время выполнения запроса без индекса / время выполнения запроса с индексом).

Таблица 4 – Сравнение результатов

Размер, Номер запроса	1	2	3	4	5	6	7	8
1 000 000	3.8851	4.4921	1.0297	5.3888	1.036	1.0234	0.9922	52.2260

Разработанный код см. в приложении А.

Выводы.

При выполнении лабораторной работы были сгенерированы тестовые данные для каждой таблицы, размерами 100, 1000, 10000, 100000, 1000000 записей в каждой таблице. Были измерены времена выполнения запросов без сортировки и с ней, добавлены индекс и проведены замеры с ними. Выяснено, что с индексами в некоторых запросах время их выполнения уменьшается, в некоторых не изменяется.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Название файла: main.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from create import Base
from insert import fill_database_base_data, fill_database_generate_data
from query import *
import argparse

DATABASE_URL = "postgresql://sun:1111@localhost:5432/work_hunter"

def create_tables(engine):
    try:
        Base.metadata.create_all(engine)
    except Exception as e:
        print(f"Error with create: {e}")

def drop_tables(engine):
    try:
        Base.metadata.drop_all(engine)
    except Exception as e:
        print(f"Error with delete: {e}")

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('-d', '--delete', action='store_true')
    parser.add_argument('-c', '--create', action='store_true')
    parser.add_argument('-f', '--fill', type=int)
    parser.add_argument('--select_all', type=str)
        parser.add_argument('-q', '--query', type=int, choices=range(1, 9),
metavar='[1-8]')
    args = parser.parse_args()

    engine = create_engine(DATABASE_URL)

    if args.delete:
        drop_tables(engine)
    if args.create:
        create_tables(engine)
    if args.fill or args.query or args.select_all:
```

```
        local_session = sessionmaker(bind=engine)
        db = local_session()

        if args.fill:
            # fill_database_base_data(db)
            fill_database_generate_data(db, args.fill)
        if args.select_all:
            select_all(db, args.select_all)
        if args.query:
            execute_query(db, args.query)

        db.close()

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

Название файла: create.py

```
from sqlalchemy.orm import DeclarativeBase, relationship
from sqlalchemy import UniqueConstraint, Numeric, Column, Integer, String, Date,
ForeignKey, Enum, BigInteger
import enum

class Base(DeclarativeBase):
    pass

class EducationType(enum.Enum):
    higher = 'Высшее'
    secondary = 'Среднее'


class VacancyStatus(enum.Enum):
    active = 'Активна'
    archive = 'В архиве'
    deleted = 'Удалена'


class WritingType(enum.Enum):
    applying = 'Зачисление'
    dismissal = 'Увольнение'
    transition = 'Переход'
    change_position = 'Изменение должности'
```

```
class Passport(Base):
    __tablename__ = 'passport'

    passport_id = Column(Integer, primary_key=True, autoincrement=True)
    passport_number = Column(Integer, nullable=False)
    passport_series = Column(Integer, nullable=False)

    __table_args__ = (
        UniqueConstraint('passport_number', 'passport_series'),
    )

    worker = relationship('Worker', back_populates='passport', uselist=False)

    def __repr__(self):
                    return f'Passport (id: {self.passport_id}, number:
{self.passport_number}, series: {self.passport_series})'


class Tin(Base):
    __tablename__ = 'tin'

    tin_id = Column(Integer, primary_key=True, autoincrement=True)
    tin = Column(BigInteger, unique=True, nullable=False)

    worker = relationship('Worker', back_populates='tin', uselist=False)

    def __repr__(self):
        return f'TIN (id: {self.tin_id}, number: {self.tin})'


class Workbook(Base):
    __tablename__ = 'workbook'

    workbook_id = Column(Integer, primary_key=True, autoincrement=True)
    workbook_number = Column(Integer, unique=True, nullable=False)
    issue_date = Column(Date, nullable=False)

    worker = relationship('WorkerWorkbook', back_populates='workbook')
            workbook_writings = relationship('WorkbookEmployerWriting',
back_populates='workbook')

    def __repr__(self):
```

```
                    return f'Workbook (id: {self.workbook.id}, number:
{self.workbook_number}, issue_date: {self.issue_date})'


class Worker(Base):
    __tablename__ = 'worker'

    worker_id = Column(Integer, primary_key=True, autoincrement=True)
          passport_id = Column(Integer, ForeignKey('passport.passport_id'),
nullable=False)
    tin_id = Column(Integer, ForeignKey('tin.tin_id'), nullable=False)
    birthday = Column(Date)

    passport = relationship('Passport', back_populates='worker')
    tin = relationship('Tin', back_populates='worker')
    workbooks = relationship('WorkerWorkbook', back_populates='worker')
    educations = relationship('WorkerEducation', back_populates='worker')
    resumes = relationship('Resume', back_populates='worker')

    def __repr__(self):
              return f'Worker (id: {self.worker_id}, tin_id: {self.tin_id},
passport_id: {self.passport.passport_id})'


class WorkerWorkbook(Base):
    __tablename__ = 'worker_workbook'

             worker_id = Column(Integer, ForeignKey('worker.worker_id'),
primary_key=True)
          workbook_id = Column(Integer, ForeignKey('workbook.workbook_id'),
primary_key=True)

    workbook = relationship('Workbook', back_populates='worker')
    worker = relationship('Worker', back_populates='workbooks')

    def __repr__(self):
            return f'WorkerWorkbook (worker_id: {self.worker_id}, workbook_id:
{self.workbook_id})'


class Education(Base):
    __tablename__ = 'education'
```

```
    education_id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String(255), nullable=False)
    type = Column(Enum(EducationType), nullable=False)

    workers = relationship('WorkerEducation', back_populates='education')

    def __repr__(self):
          return f'Education (id: {self.education_id}, name: {self.name}, type:
{self.type})'


class WorkerEducation(Base):
    __tablename__ = 'worker_education'

              worker_id   =   Column(Integer,   ForeignKey('worker.worker_id'),
primary_key=True)
         education_id  =  Column(Integer,  ForeignKey('education.education_id'),
primary_key=True)
    enroll_date = Column(Date, nullable=False)

    worker = relationship('Worker', back_populates='educations')
    education = relationship('Education', back_populates='workers')

    def __repr__(self):
           return f'WorkerEducation (worker_id: {self.worker_id}, education_id:
{self.education_id}, enroll_date: {self.enroll_date})'


class Post(Base):
    __tablename__ = 'post'

    post_id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String(255), unique=True, nullable=False)

    vacancies = relationship('Vacancy', back_populates='post')
    resumes = relationship('Resume', back_populates='post')

    def __repr__(self):
        return f'Post (id: {self.post_id}, name: {self.name})'


class Resume(Base):
    __tablename__ = 'resume'
```

```
    resume_id = Column(Integer, primary_key=True, autoincrement=True)
    worker_id = Column(Integer, ForeignKey('worker.worker_id'), nullable=False)
    post_id = Column(Integer, ForeignKey('post.post_id'), nullable=False)
    salary = Column(Numeric(8,2), nullable=False)

    worker = relationship('Worker', back_populates='resumes')
    post = relationship('Post', back_populates='resumes')
    skills = relationship('ResumeSkill', back_populates='resumes')

    def __repr__(self):
            return f'Resume (id: {self.resume_id}, worker_id: {self.worker_id},
post_id: {self.post_id}, salary: {self.salary})'


class Skill(Base):
    __tablename__ = 'skill'

    skill_id = Column(Integer, primary_key=True, autoincrement=True)
    description = Column(String(255), unique=True, nullable=False)

    resumes = relationship('ResumeSkill', back_populates='skills')

    def __repr__(self):
        return f'Skill (id: {self.skill_id}, descriprion: {self.description})'


class ResumeSkill(Base):
    __tablename__ = 'resume_skill'

                resume_id    =    Column(Integer,    ForeignKey('resume.resume_id'),
primary_key=True)
    skill_id = Column(Integer, ForeignKey('skill.skill_id'), primary_key=True)

    resumes = relationship('Resume', back_populates='skills')
    skills = relationship('Skill', back_populates='resumes')

    def __repr__(self):
                return  f'ResumeSkill  (resume_id:  {self.resume_id},  skill_id:
{self.skill_id})'


class FieldOfActivity(Base):
```

```
    __tablename__ = 'field_of_activity'

    activity_id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String(255), unique=True, nullable=False)

    employers = relationship('Employer', back_populates='field_of_activity')

    def __repr__(self):
        return f'FieldOfActivity (id: {self.activity_id}, name: {self.name})'


class Employer(Base):
    __tablename__ = 'employer'

    employer_id = Column(Integer, primary_key=True, autoincrement=True)
      activity_id = Column(Integer, ForeignKey('field_of_activity.activity_id'),
nullable=False)
    name = Column(String(255), nullable=False)
    photo_url = Column(String(255))
    description = Column(String(255))

                    field_of_activity     =     relationship('FieldOfActivity',
back_populates='employers')
                   presentation_files     =     relationship('PresentationFile',
back_populates='employer')
    vacancies = relationship('Vacancy', back_populates='employer')
               workbook_writings    =    relationship('WorkbookEmployerWriting',
back_populates='employer')

    def __repr__(self):
                    return   f'Employer   (id:   {self.employer_id},   activity_id:
{self.activity_id},   name:   {self.name}),   name:   {self.name},   photo_url:
{self.photo_url}, description: {self.description})'


class PresentationFile(Base):
    __tablename__ = 'presentation_file'

    file_id = Column(Integer, primary_key=True, autoincrement=True)
          employer_id   =   Column(Integer,   ForeignKey('employer.employer_id'),
nullable=False)
    name = Column(String(255), nullable=False)
    url = Column(String(255), nullable=False)
```

```
    employer = relationship('Employer', back_populates='presentation_files')

    def __repr__(self):
                return f'PresentationFile (id: {self.file_id}, employer_id:
{self.employer_id}, name: {self.name}, url: {self.url})'


class WorkbookWriting(Base):
    __tablename__ = 'workbook_writing'

    writing_id = Column(Integer, primary_key=True, autoincrement=True)
    type = Column(Enum(WritingType), nullable=False)
    date = Column(Date, nullable=False)
    description = Column(String(255))

                workbook_writings    =    relationship('WorkbookEmployerWriting',
back_populates='workbook_writings')

    def __repr__(self):
            return f'WorkbookWriting (id: {self.writing_id}, type: {self.type},
date: {self.date}, description: {self.description})'


class WorkbookEmployerWriting(Base):
    __tablename__ = 'workbook_employer_writing'

       writing_id = Column(Integer, ForeignKey('workbook_writing.writing_id'),
primary_key=True)
           workbook_id   =   Column(Integer,   ForeignKey('workbook.workbook_id'),
primary_key=True)
           employer_id   =   Column(Integer,   ForeignKey('employer.employer_id'),
primary_key=True)

                    workbook_writings     =     relationship('WorkbookWriting',
back_populates='workbook_writings')
    employer = relationship('Employer', back_populates='workbook_writings')
    workbook = relationship('Workbook', back_populates='workbook_writings')

    def __repr__(self):
              return f'WorkbookEmployerWriting (writing_id: {self.writing_id},
workbook_id: {self.workbook_id}, employer_id: {self.employer_id})'
```

```
class Vacancy(Base):
    __tablename__ = 'vacancy'

    vacancy_id = Column(Integer, primary_key=True, autoincrement=True)
    post_id = Column(Integer, ForeignKey('post.post_id'), nullable=False)
          employer_id  =  Column(Integer,  ForeignKey('employer.employer_id'),
nullable=False)
    name = Column(String(255), nullable=False)
    address = Column(String(255), nullable=False)
    salary = Column(Numeric(8,2), nullable=False)
    experience = Column(Integer)
    comment = Column(String(255))
    status = Column(Enum(VacancyStatus), nullable=False)

    post = relationship('Post', back_populates='vacancies')
    employer = relationship('Employer', back_populates='vacancies')
                    requirements     =     relationship('VacancyRequirement',
back_populates='vacancies')
    conditions = relationship('VacancyCondition', back_populates='vacancies')

    def __repr__(self):
            return  f'Vacancy  (id:  {self.vacancy_id},  post_id:  {self.post_id},
employer_id:  {self.employer_id},  name:  {self.name},  address:  {self.address},
salary: {self.salary}, experience: {self.experience}, comment: {self.comment},
status: {self.status})'


class Requirement(Base):
    __tablename__ = 'requirement'

    requirement_id = Column(Integer, primary_key=True, autoincrement=True)
    description = Column(String(255), unique=True, nullable=False)

                        vacancies     =     relationship('VacancyRequirement',
back_populates='requirements')

    def __repr__(self):
               return  f'Requirement  (id:  {self.requirement_id},  description:
{self.description})'


class VacancyRequirement(Base):
```

```
    __tablename__ = 'vacancy_requirement'

            vacancy_id   =   Column(Integer,   ForeignKey('vacancy.vacancy_id'),
primary_key=True)
     requirement_id = Column(Integer, ForeignKey('requirement.requirement_id'),
primary_key=True)

    vacancies = relationship('Vacancy', back_populates='requirements')
    requirements = relationship('Requirement', back_populates='vacancies')

    def __repr__(self):
        return f'VacancyRequirement (vacancy_id: {self.vacancy_id}, requirement:
{self.requirement})'


class Condition(Base):
    __tablename__ = 'condition'

    condition_id = Column(Integer, primary_key=True, autoincrement=True)
    description = Column(String(255), unique=True, nullable=False)

    vacancies = relationship('VacancyCondition', back_populates='conditions')

    def __repr__(self):
                return  f'Condition  (id:  {self.condition_id},  description:
{self.description})'


class VacancyCondition(Base):
    __tablename__ = 'vacancy_condition'

            vacancy_id   =   Column(Integer,   ForeignKey('vacancy.vacancy_id'),
primary_key=True)
        condition_id  =  Column(Integer,  ForeignKey('condition.condition_id'),
primary_key=True)

    vacancies = relationship('Vacancy', back_populates='conditions')
    conditions = relationship('Condition', back_populates='vacancies')

    def __repr__(self):
          return f'VacancyCondition (vacancy_id: {self.vacancy_id}, condition:
{self.condition_id})'
```

Название файла: insert.py

```
from datetime import date
from generate import *

passports = [
    Passport(passport_number=3317, passport_series=627219),
    Passport(passport_number=3214, passport_series=745323),
    Passport(passport_number=3421, passport_series=231421),
    Passport(passport_number=3142, passport_series=892156),
    Passport(passport_number=3345, passport_series=324561),
    Passport(passport_number=3214, passport_series=234741)
]

tins = [
    Tin(tin=505021345692),
    Tin(tin=432102345672),
    Tin(tin=476221556514),
    Tin(tin=813267892103),
    Tin(tin=456969651393),
    Tin(tin=158923546790)
]

workers = [
    Worker(passport_id=2, tin_id=3, birthday=date(1992, 2, 14)),
    Worker(passport_id=1, tin_id=4, birthday=date(1996, 4, 19)),
    Worker(passport_id=5, tin_id=6, birthday=date(1982, 10, 1)),
    Worker(passport_id=6, tin_id=1, birthday=date(2003, 1, 27)),
    Worker(passport_id=3, tin_id=5, birthday=date(1979, 7, 8)),
    Worker(passport_id=4, tin_id=2, birthday=date(2000, 1, 1))
]

educations = [
    Education(name='СПБГЭТУ',   type=EducationType.higher),
    Education(name='МАИ',       type=EducationType.higher),
    Education(name='КЛПК',      type=EducationType.secondary),
    Education(name='ВятГУ',     type=EducationType.higher),
    Education(name='ИТМО',      type=EducationType.higher),
    Education(name='РГГУ',      type=EducationType.secondary),
    Education(name='КПиАС',     type=EducationType.secondary),
    Education(name='СПБГУГА',   type=EducationType.higher)
]
```

```
worker_educations = [
    WorkerEducation(worker_id=1, education_id=3, enroll_date=date(2010, 9, 1)),
    WorkerEducation(worker_id=1, education_id=1, enroll_date=date(2015, 8, 25)),
    WorkerEducation(worker_id=3, education_id=4, enroll_date=date(2012, 7, 29)),
    WorkerEducation(worker_id=5, education_id=8, enroll_date=date(2000, 8, 27)),
    WorkerEducation(worker_id=5, education_id=5, enroll_date=date(1992, 8, 28)),
    WorkerEducation(worker_id=4, education_id=2, enroll_date=date(2019, 9, 1)),
    WorkerEducation(worker_id=6, education_id=1, enroll_date=date(2018, 8, 23))
]

workbooks = [
    Workbook(workbook_number=8627821, issue_date=date(2006, 12, 14)),
    Workbook(workbook_number=5423109, issue_date=date(2001, 3, 27)),
    Workbook(workbook_number=1234901, issue_date=date(1997, 9, 2)),
    Workbook(workbook_number=4298134, issue_date=date(2015, 2, 13)),
    Workbook(workbook_number=9823518, issue_date=date(2020, 7, 9)),
    Workbook(workbook_number=7349821, issue_date=date(2025, 1, 1)),
    Workbook(workbook_number=2354890, issue_date=date(2017, 9, 18))
]

worker_workbooks = [
    WorkerWorkbook(worker_id=1, workbook_id=1),
    WorkerWorkbook(worker_id=3, workbook_id=2),
    WorkerWorkbook(worker_id=6, workbook_id=5),
    WorkerWorkbook(worker_id=5, workbook_id=3),
    WorkerWorkbook(worker_id=4, workbook_id=6),
    WorkerWorkbook(worker_id=5, workbook_id=4),
    WorkerWorkbook(worker_id=2, workbook_id=7)
]

posts = [
    Post(name='Программист на python'),
    Post(name='Курьер'),
    Post(name='Складовщик'),
    Post(name='HR-менеджер'),
    Post(name='Кассир'),
    Post(name='Водитель автобуса'),
    Post(name='Маркетолог'),
    Post(name='Data-Scientist'),
    Post(name='Парикмахер'),
    Post(name='Разнорабочий')
]
```

```
skills = [
    Skill(description='Умение работать в команде'),
    Skill(description='Водительская категория D'),
    Skill(description='Программирование на python'),
    Skill(description='Мат. статистика'),
    Skill(description='Трудолюбие'),
    Skill(description='Терпеливость'),
    Skill(description='Быстрая адаптация к изменениям'),
    Skill(description='Умение работать в стрессовой обстановке'),
    Skill(description='Спортивное физическое телосложение')
]

resumes = [
    Resume(worker_id=1, post_id=4,  salary=75000),
    Resume(worker_id=5, post_id=6,  salary=57500.50),
    Resume(worker_id=3, post_id=3,  salary=70000),
    Resume(worker_id=3, post_id=10, salary=47000.99),
    Resume(worker_id=4, post_id=8,  salary=250000),
    Resume(worker_id=6, post_id=1,  salary=150000),
    Resume(worker_id=5, post_id=2,  salary=180000)
]

resume_skills = [
    ResumeSkill(resume_id=1, skill_id=7),
    ResumeSkill(resume_id=1, skill_id=8),
    ResumeSkill(resume_id=1, skill_id=1),
    ResumeSkill(resume_id=5, skill_id=4),
    ResumeSkill(resume_id=6, skill_id=3),
    ResumeSkill(resume_id=7, skill_id=5),
    ResumeSkill(resume_id=7, skill_id=7),
    ResumeSkill(resume_id=2, skill_id=2),
    ResumeSkill(resume_id=3, skill_id=9),
    ResumeSkill(resume_id=3, skill_id=5)
]

field_of_activitys = [
    FieldOfActivity(name='Перевозка пассажиров'),
    FieldOfActivity(name='IT'),
    FieldOfActivity(name='Доставка'),
    FieldOfActivity(name='Продажи'),
    FieldOfActivity(name='Производство продукции'),
]
```

```
employers = [
                        Employer(activity_id=2,      name='КиберПредсказания',
photo_url='https://photo1',
            description='Компания, занимающаяся аналитикой данных, от сторонних
организаций'),
                            Employer(activity_id=3,       name='ЯнлексПитье',
photo_url='https://photo2',
            description='Служба доставки питья на дом'),
                        Employer(activity_id=1,      name='ОАО      "Везем"',
photo_url='https://photo3',
               description='Организация, осуществляющая перевозку пассажиров по
маршрутам дальнего следования'),
                            Employer(activity_id=5,       name='SuperFlowers',
photo_url='https://photo4',
            description='Производство блоков питания'),
                  Employer(activity_id=4,     name='ООО     "Купи     Слона"',
photo_url='https://photo5',
            description='Зоомагазин'),
                                Employer(activity_id=5,        name='ОкноДом',
photo_url='https://photo6',
            description='Отечественный производитель стеклопакетов')
]

presentation_files = [
    PresentationFile(employer_id=1, name='Презентация 1', url='https://file1'),
    PresentationFile(employer_id=1, name='Презентация 2', url='https://file2'),
    PresentationFile(employer_id=2, name='Презентация 1', url='https://file3'),
    PresentationFile(employer_id=5, name='Презентация 1', url='https://file4'),
    PresentationFile(employer_id=4, name='Презентация 1', url='https://file5')
]

requirements = [
    Requirement(description='Стрессоустойчивость'),
    Requirement(description='Знание ПДД'),
    Requirement(description='Знание python на базовом уровне'),
    Requirement(description='Знание python на повышенном уровне'),
    Requirement(description='Знание мат. статистики'),
    Requirement(description='Работа в команде'),
    Requirement(description='Хорошо поставленная речь'),
    Requirement(description='Наличие автомобиля'),
    Requirement(description='Работа из офиса'),
    Requirement(description='Работа из дома')
]
```

```
conditions = [
    Condition(description='Обустроенное рабочее место'),
    Condition(description='Предоставление ноутбука'),
    Condition(description='ДМС'),
    Condition(description='Гибкий рабочий график'),
    Condition(description='График работы 2 на 2'),
    Condition(description='Работа в большой компании'),
    Condition(description='Официальное трудоустройство'),
    Condition(description='Рабочий автомобиль от компании')
]

vacancies = [
    Vacancy(post_id=8, employer_id=1, name='DataScientist для Предсказаний',
                           address='Варшавская  улица,  63к1,  Санкт-Петербург',
salary=180000.50,   experience=5,
              comment='Требуется DataScientist для предсказаний кибер будущего',
status=VacancyStatus.active),
    Vacancy(post_id=6, employer_id=3, name='Водитель рейсового автобуса',
               address='улица Костюшко, 17, Санкт-Петербург',       salary=83250,
experience=3,
            status=VacancyStatus.archive),
    Vacancy(post_id=4, employer_id=1, name='Менеджер по отбору кадров (HR)',
               address='Варшавская улица, 44, Санкт-Петербург',     salary=91000,
experience=7,
                comment='Нужен менеджер для подбора персонала в новую команду',
status=VacancyStatus.active),
    Vacancy(post_id=2, employer_id=2, name='Курьер доставки питья',
            address='Варшавская улица, 124, Санкт-Петербург',   salary=110000,
            status=VacancyStatus.active),
    Vacancy(post_id=3, employer_id=6, name='Начальник склада',
               address='Софийская улица, 8к5с4, Санкт-Петербург',  salary=87000,
experience=5,
             comment='Начальник-заведующий складом, управляет приемом и отправкой
товара', status=VacancyStatus.archive),
    Vacancy(post_id=3, employer_id=6, name='Складовщик на новый склад',
            address='Софийская улица, 8к5с4, Санкт-Петербург',  salary=63000,
            status=VacancyStatus.active),
    Vacancy(post_id=7, employer_id=5, name='Маркетолог бренда',
            address='Московский проспект, 9, Санкт-Петербург',  salary=69000.99,
                    comment='Ждем креативных людей для продвижения бренда!',
status=VacancyStatus.active)
]
```

```
vacancy_requirements = [
    VacancyRequirement(vacancy_id=1, requirement_id=3),
    VacancyRequirement(vacancy_id=1, requirement_id=4),
    VacancyRequirement(vacancy_id=3, requirement_id=6),
    VacancyRequirement(vacancy_id=3, requirement_id=7),
    VacancyRequirement(vacancy_id=7, requirement_id=9),
    VacancyRequirement(vacancy_id=4, requirement_id=8),
    VacancyRequirement(vacancy_id=5, requirement_id=1),
    VacancyRequirement(vacancy_id=5, requirement_id=7),
    VacancyRequirement(vacancy_id=2, requirement_id=2),
    VacancyRequirement(vacancy_id=1, requirement_id=10)
]

vacancy_conditions = [
    VacancyCondition(vacancy_id=1, condition_id=2),
    VacancyCondition(vacancy_id=1, condition_id=4),
    VacancyCondition(vacancy_id=2, condition_id=5),
    VacancyCondition(vacancy_id=2, condition_id=8),
    VacancyCondition(vacancy_id=6, condition_id=3),
    VacancyCondition(vacancy_id=7, condition_id=6),
    VacancyCondition(vacancy_id=4, condition_id=7),
    VacancyCondition(vacancy_id=5, condition_id=1)
]

workbook_writings = [
    WorkbookWriting(type=WritingType.applying,          date=date(2010, 1, 1)),
     WorkbookWriting(type=WritingType.change_position,    date=date(2018, 2, 13),
description='Повышение должности'),
    WorkbookWriting(type=WritingType.applying,          date=date(1995, 7, 1)),
      WorkbookWriting(type=WritingType.dismissal,            date=date(1995, 11,
18)),
    WorkbookWriting(type=WritingType.applying,          date=date(1995, 12, 1)),
    WorkbookWriting(type=WritingType.dismissal,         date=date(1997, 5, 10)),
    WorkbookWriting(type=WritingType.applying,          date=date(1998, 1, 13)),
    WorkbookWriting(type=WritingType.dismissal,         date=date(2003, 3, 8)),
    WorkbookWriting(type=WritingType.applying,          date=date(2004, 5, 21)),
    WorkbookWriting(type=WritingType.applying,          date=date(2023, 7, 18))
]

workbook_employer_writings = [
    WorkbookEmployerWriting(writing_id=1,   workbook_id=1, employer_id=6),
    WorkbookEmployerWriting(writing_id=2,   workbook_id=1, employer_id=6),
```

```
    WorkbookEmployerWriting(writing_id=4,   workbook_id=4, employer_id=2),
    WorkbookEmployerWriting(writing_id=3,   workbook_id=4, employer_id=2),
    WorkbookEmployerWriting(writing_id=10,  workbook_id=6, employer_id=5),
    WorkbookEmployerWriting(writing_id=5,   workbook_id=4, employer_id=4),
    WorkbookEmployerWriting(writing_id=6,   workbook_id=4, employer_id=4),
    WorkbookEmployerWriting(writing_id=8,   workbook_id=3, employer_id=3),
    WorkbookEmployerWriting(writing_id=7,   workbook_id=3, employer_id=3),
    WorkbookEmployerWriting(writing_id=9,   workbook_id=3, employer_id=1),
]

def fill_database_base_data(db):
    db.add_all(passports)
    db.add_all(tins)
    db.add_all(workers)
    db.commit()

    db.add_all(educations)
    db.add_all(worker_educations)
    db.commit()

    db.add_all(posts)
    db.commit()

    db.add_all(skills)
    db.add_all(resumes)
    db.add_all(resume_skills)
    db.commit()

    db.add_all(field_of_activitys)
    db.add_all(employers)
    db.add_all(presentation_files)
    db.commit()

    db.add_all(workbooks)
    db.add_all(worker_workbooks)
    db.commit()

    db.add_all(workbook_writings)
    db.add_all(workbook_employer_writings)
    db.commit()

    db.add_all(vacancies)
    db.commit()
```

```
    db.add_all(requirements)
    db.add_all(vacancy_requirements)
    db.commit()

    db.add_all(conditions)
    db.add_all(vacancy_conditions)
    db.commit()


def fill_database_generate_data(db, count):
    new_passports = generate_passports(count)
    db.add_all(new_passports)
    db.commit()
    print('insert passport')

    new_tins = generate_tins(count)
    db.add_all(new_tins)
    db.commit()
    print('insert tin')

    new_educations = generate_educations(count)
    db.add_all(new_educations)
    db.commit()
    print('insert education')

    new_workers = generate_workers(count, new_passports, new_tins)
    db.add_all(new_workers)
    db.commit()
    print('insert worker')

            new_worker_educations   =   generate_worker_educations(new_workers,
new_educations)
    db.add_all(new_worker_educations)
    db.commit()
    print('insert worker-education')

    new_posts = generate_posts(count)
    db.add_all(new_posts)
    db.commit()
    print('insert post')

    new_skills = generate_skills(count)
```

```
    db.add_all(new_skills)
    db.commit()
    print('insert skill')

    new_resumes = generate_resumes(new_workers, new_posts)
    db.add_all(new_resumes)
    db.commit()
    print('insert resume')

    new_resume_skills = generate_resume_skills(new_resumes, new_skills)
    db.add_all(new_resume_skills)
    db.commit()
    print('insert resume-skill')

    new_field_of_activitys = generate_field_of_activities(count)
    db.add_all(new_field_of_activitys)
    db.commit()
    print('insert field-of-activity')

    new_employers = generate_employers(count, new_field_of_activitys)
    db.add_all(new_employers)
    db.commit()
    print('insert employer')

    new_presentation_files = generate_presentation_file(count, new_employers)
    db.add_all(new_presentation_files)
    db.commit()
    print('insert presentation-file')

    new_workbooks = generate_workbooks(count)
    db.add_all(new_workbooks)
    db.commit()
    print('insert workbook')

        new_worker_workbooks = generate_worker_workbooks(count, new_workers,
new_workbooks)
    db.add_all(new_worker_workbooks)
    db.commit()
    print('insert worker-workbook')

    new_workbook_writings = generate_workbook_writings(count)
    db.add_all(new_workbook_writings)
    db.commit()
```

```
    print('insert writing')

                                        new_workbook_employer_writings          =
generate_workbook_employer_writings(new_workbook_writings,        new_workbooks,
new_employers)
    db.add_all(new_workbook_employer_writings)
    db.commit()
    print('insert workbook-employer-writing')

    new_vacancies = generate_vacancies(count, new_posts, new_employers)
    db.add_all(new_vacancies)
    db.commit()
    print('insert vacancy')

    new_requirements = generate_requirements(count)
    db.add_all(new_requirements)
    db.commit()
    print('insert requirement')

      new_vacancy_requirements  =  generate_vacancy_requirements(new_requirements,
new_vacancies)
    db.add_all(new_vacancy_requirements)
    db.commit()
    print('insert vacancy-requirement')

    new_conditions = generate_conditions(count)
    db.add_all(new_conditions)
    db.commit()
    print('insert condition')

          new_vacancy_conditions   =   generate_vacancy_conditions(new_conditions,
new_vacancies)
    db.add_all(new_vacancy_conditions)
    db.commit()
    print('insert vacancy-condition')
```

Название файла: providers.py

```
from faker.providers import BaseProvider


class EducationNameProvider(BaseProvider):
    def education_name(self):
```

```
        templates = [
            '{city} {type} {prefix} {specialization} №{number} имени {name}',
            '{prefix} {city} {type} {specialization} №{number} имени {name}',
            '{type} {specialization} в {city} №{number} имени {name}',
            '{city} {specialization} {type} №{number} имени {name}',
            '{prefix} {type} {specialization} №{number} имени {name}',
            '{specialization} {type} имени {name} №{number}',
            '{city} {type} №{number} имени {name}'
        ]

        #3
        types = ['Университет', 'Институт', 'Колледж']

        #20
              cities = ['Московский', 'Санкт-Петербургский', 'Новосибирский',
'Екатеринбургский',
                        'Казанский', 'Уральский', 'Сибирский', 'Дальневосточный',
'Воронежский',
                    'Ростовский', 'Краснодарский', 'Нижегородский', 'Самарский',
'Омский',
                            'Вятский', 'Мурманский', 'Ижевский', 'Сыктывкарский',
'Красноярский', 'Магнитогорский']

        #11
                prefixes = ['государственный', 'федеральный', 'национальный',
'технический',
                                'медицинский', 'педагогический', 'экономический',
'юридический',
                                            'политехнический', 'исследовательский',
'электро-технический']
        #15
        specializations = [
            'имени Ломоносова', 'имени Баумана', 'технологий и дизайна',
                        'международных отношений', 'информационных технологий',
'строительный',
            'транспортный', 'энергетический', 'пищевой промышленности',
                    'легкой промышленности', 'культуры и искусств', 'физической
культуры',
            'управления', 'сервиса', 'туризма и гостеприимства'
        ]

        #11
```

```
            names = ['Ломоносова', 'Баумана', 'Менделеева', 'Циолковского',
'Королева',
                     'Попова', 'Вавилова', 'Вернадского', 'Губкина', 'Плеханова',
'Ленина']

        template = self.random_element(templates)

        result = template.format(
            type=self.random_element(types),
            city=self.random_element(cities),
            prefix=self.random_element(prefixes),
            specialization=self.random_element(specializations),
            name=self.random_element(names),
            number=self.random_int(min=1, max=15)
        )

        return result


class SkillRequirementProvider(BaseProvider):
    def skill_requirement_description(self):
        templates = [
            'Профессиональные навыки: {technical}, {tool} {level}, {technology},
{subject}',
                'Ключевые компетенции: работа {context}, {quality}, {personal}
качество',
                    'Квалификация: {certification}, знание {subject}, опыт с
{technology}',
            'Требования: {technical}, {tool} {level}, {context}',
            'Компетенции: {personal} мышление, {quality}, {technical} {level}'
            'Умение работать {context}',
                    'Навыки {technical} {level} и знание {subject}, опыт с
{technology}',
            '{quality} как профессиональное качество',
            'Владение {tool}',
            'Сертификат {certification}',
               'Опыт работы с {technology} {level}, навыки {technical} и знание
{subject}',
            'Знание {subject} {level}, умение работать {context}',
            '{personal} качество',
            'Умение работать {context} и владение {tool} {level}',
            'Навыки {technical} {level} и знание {subject}',
            '{quality} и {personal} качество',
```

```
            'Владение {tool} {level} и опыт работы с {technology}',
            'Сертификат {certification} и умение работать {context}',
            'Опыт работы с {technology} {level} и навыки {technical}',
            'Знание {subject} и сертификат {certification}',
            '{personal} качество и владение {tool} {level} {context}',
        ]

        #12
        contexts = [
            'в команде', 'удаленно', 'в стрессовой обстановке', 'с клиентами',
             'в международной среде', 'в условиях неопределенности', 'с большими
объемами данных',
                  'в agile-среде', 'с распределенными командами', 'в режиме
многозадачности',
            'с конфиденциальной информацией', 'в быстро меняющейся среде'
        ]

        #23
        technicals = [
                 'программирования', 'анализа данных', 'управления проектами',
'ведения документации',
                'машинного обучения', 'веб-разработки', 'мобильной разработки',
'кибербезопасности',
            'тестирования ПО', 'DevOps практик', 'системного администрирования',
'сетевых технологий',
              'баз данных', 'искусственного интеллекта', 'блокчейн технологий',
'облачных вычислений',
                  'водительская категория D', 'готовки', 'уборки', 'продажи',
'менеджмента',
            'рисования', 'чистки', 'управления', 'маркетинга'
        ]
        #12
        qualities = [
            'Трудолюбие', 'Терпеливость', 'Ответственность', 'Внимательность',
            'Инициативность', 'Адаптивность', 'Креативность', 'Настойчивость',
                     'Дисциплинированность', 'Пунктуальность', 'Аккуратность',
'Стрессоустойчивость'
        ]
        #27
        tools = [
                    'Python', 'SQL', 'Excel', 'Photoshop', 'Git', 'Docker',
'Kubernetes',
```

```
                'AWS', 'Azure', 'Google Cloud', 'JIRA', 'Confluence', 'Figma',
'Sketch',
            'Tableau', 'Power BI', 'React', 'Angular', 'Vue', 'Node.js', 'Java',
            'C++', 'C#', 'PHP', 'Ruby', 'Swift', 'Kotlin'
        ]
        #16
        certifications = [
                    'PMI', 'Scrum Master', 'водительская категория D', 'первой
медицинской помощи',
                'AWS Certified', 'Google Professional', 'Microsoft Certified',
'CISSP',
            'CEH', 'CISA', 'ITIL', 'Six Sigma', 'TOGAF', 'Prince2', 'CPA', 'CFA'
        ]
        #14
        technologies = [
               'веб-технологиями', 'базами данных', 'искусственным интеллектом',
'облачными решениями',
                   'микросервисной архитектурой', 'контейнеризацией', 'большими
данными', 'IoT',
                    'компьютерным зрением', 'обработкой естественного языка',
'робототехникой',
               'виртуальной реальностью', 'дополненной реальностью', 'квантовыми
вычислениями'
        ]
        #16
        subjects = [
                    'математической статистики', 'экономики', 'юриспруденции',
'иностранных языков',
                        'линейной алгебры', 'математического анализа', 'теории
вероятностей', 'финансов',
                'бухгалтерского учета', 'менеджмента', 'маркетинга', 'психологии',
'социологии',
            'философии', 'истории', 'культурологии'
        ]
        #12
        personal_qualities = [
            'Лидерское', 'Коммуникативное', 'Аналитическое', 'Творческое',
            'Стратегическое', 'Критическое', 'Системное', 'Логическое',
            'Техническое', 'Бизнес', 'Эмоциональное', 'Межкультурное'
        ]
        #8
        levels = [
```

```
                'на базовом уровне', 'на продвинутом уровне', 'на экспертном
уровне',
                  'на профессиональном уровне', 'с углубленным знанием', 'с
сертификацией',
            'с многолетним опытом', 'с подтвержденным опытом'
        ]

        template = self.random_element(templates)

        result = template.format(
            context=self.random_element(contexts),
            technical=self.random_element(technicals),
            quality=self.random_element(qualities),
            tool=self.random_element(tools),
            certification=self.random_element(certifications),
            technology=self.random_element(technologies),
            subject=self.random_element(subjects),
            personal=self.random_element(personal_qualities),
            level=self.random_element(levels)
        )

        return result


class FieldOfActivityProvider(BaseProvider):
    def field_of_activity(self):
        templates = [
            '{action} {object} и {service} услуги в {industry}',
            'Производство {product} и продажа {goods} {service}',
                 'Разработка {service} {tech} и консалтинг в области {domain}
{product}',
            '{action} {industry} и обучение {skill} {service}',
            'Обслуживание {equipment} и аренда {property} {industry}',
            '{action} {service} услуги и разработка {tech}',
            'Производство {product} с обслуживанием {equipment}',
            'Консалтинг в области {domain} и обучение {industry} {skill}',
            '{scale} {action} {object}',
            '{industry}: {action} {object} и {service} услуги',
              '{action}, {industry}, Производство {product} и разработка {tech}
{product}',
            '{scale} консалтинг в области {domain} и {industry}',
            'Обучение {skill} с {tech} разработкой'
        ]
```

```
        #25
        actions = [
                        'Перевозка', 'Доставка', 'Производство', 'Разработка',
'Обслуживание',
                    'Ремонт', 'Аренда', 'Продажа', 'Обучение', 'Консалтинг',
'Внедрение',
                     'Интеграция', 'Мониторинг', 'Аналитика', 'Оптимизация',
'Управление',
                   'Продвижение', 'Дистрибуция', 'Хранение', 'Трансформация',
'Создание',
            'Исследование', 'Тестирование', 'Аудит', 'Лизинг'
        ]
        #18
        objects = [
                   'пассажиров', 'грузов', 'товаров', 'данных', 'программного
обеспечения',
            'информации', 'документов', 'материалов', 'сырья', 'оборудования',
                    'техники', 'ценностей', 'активов', 'персональных данных',
'мультимедиа',
            'цифровых активов', 'криптовалют', 'ценных бумаг'
        ]

        #26
        industries = [
            'IT', 'Финансы', 'Строительство', 'Недвижимость', 'Медицина',
                  'Образование', 'Туризм', 'Розничная торговля', 'Маркетинг',
'Логистика',
                     'Энергетика', 'Телекоммуникации', 'Сельское хозяйство',
'Фармацевтика',
                    'Автомобильная промышленность', 'Авиация', 'Судостроение',
'Металлургия',
              'Химическая промышленность', 'Пищевая промышленность', 'Банковское
дело',
            'Страхование', 'Юриспруденция', 'Культура', 'Спорт', 'Развлечения'
        ]
        #29
        services = [
                     'банковские', 'страховые', 'юридические', 'медицинские',
'образовательные',
                    'транспортные', 'логистические', 'консалтинговые', 'IT',
'ремонтные',
```

```
                'финансовые', 'бухгалтерские', 'миграционные', 'риелторские',
'оценочные',
            'клининговые', 'охранные', 'маркетинговые', 'рекламные', 'PR', 'HR',
                            'рекрутинговые', 'аутсорсинговые', 'аналитические',
'исследовательские',
            'проектные', 'инжиниринговые', 'архитектурные', 'дизайнерские'
        ]
        #20
        products = [
                'продукции', 'оборудования', 'мебели', 'одежды', 'продуктов
питания',
                    'электроники', 'строительных материалов', 'автомобилей',
'медикаментов',
            'химической продукции', 'металлопроката', 'пластиковых изделий',
                'деревянных конструкций', 'текстиля', 'косметики', 'бытовой
техники',
            'промышленных роботов', 'упаковки', 'печатной продукции', 'ювелирных
изделий'
        ]
        #20
        tech = [
            'программного обеспечения', 'мобильных приложений', 'веб-сайтов',
            'искусственного интеллекта', 'баз данных', 'кибербезопасности',
                'облачных решений', 'блокчейн технологий', 'IoT систем', 'AR/VR
приложений',
            'компьютерного зрения', 'машинного обучения', 'больших данных',
            'микросервисных архитектур', 'DevOps инструментов', 'CRM систем',
            'ERP систем', 'SCM систем', 'BI решений', 'chatbot платформ'
        ]
        #16
        equipment = [
            'оборудования', 'техники', 'автомобилей', 'компьютеров', 'серверов',
                'сетевого оборудования', 'медицинской техники', 'промышленных
станков',
                'строительной техники', 'оргтехники', 'телекоммуникационного
оборудования',
                    'лабораторного оборудования', 'кухонного оборудования',
'холодильного оборудования',
            'климатического оборудования', 'энергетического оборудования'
        ]
        #19
        domains = [
            'менеджмента', 'финансов', 'IT', 'маркетинга', 'юриспруденции',
```

```
            'безопасности', 'логистики', 'HR', 'продаж', 'закупок',
            'производства', 'качества', 'инноваций', 'устойчивого развития',
                        'цифровой трансформации', 'управления проектами',
'риск-менеджмента',
            'стратегического планирования', 'корпоративного управления'
        ]
        #16
        goods = [
            'товаров', 'продукции', 'услуг', 'недвижимости', 'автомобилей',
            'акций', 'облигаций', 'валюты', 'криптовалют', 'сырья',
            'энергоресурсов', 'интеллектуальной собственности', 'франшиз',
            'лицензий', 'патентов', 'дистрибьюторских прав'
        ]

        property = [
            'недвижимости', 'транспорта', 'оборудования', 'техники',
            'коммерческой недвижимости', 'жилой недвижимости', 'спецтехники',
            'воздушных судов', 'морских судов', 'железнодорожного состава',
            'строительной техники', 'складских помещений', 'офисных помещений'
        ]
        #13
        skills = [
                    'программированию', 'менеджменту', 'маркетингу', 'дизайну',
'финансам',
                        'аналитике', 'продажам', 'переговорам', 'лидерству',
'коммуникациям',
            'иностранным языкам', 'цифровой грамотности', 'кибербезопасности',
                'управлению проектами', 'бережливому производству', 'аггильным
методологиям',
              'бизнес-анализу', 'дата-сайенс', 'DevOps практикам', 'тестированию
ПО'
        ]
        #12
        scales = [
            'Международное', 'Федеральное', 'Региональное', 'Локальное',
            'Корпоративное', 'Частное', 'Государственное', 'Муниципальное',
            'Крупномасштабное', 'Среднее', 'Маломасштабное', 'Стартап'
        ]

        template = self.random_element(templates)

        result = template.format(
            action=self.random_element(actions),
```

```
            object=self.random_element(objects),
            industry=self.random_element(industries),
            service=self.random_element(services),
            product=self.random_element(products),
            tech=self.random_element(tech),
            equipment=self.random_element(equipment),
            domain=self.random_element(domains),
            goods=self.random_element(goods),
            property=self.random_element(property),
            skill=self.random_element(skills),
            scale=self.random_element(scales),
        )

        return result


class ConditionProvider(BaseProvider):
    def condition_description(self):
        templates = [
            '{work_conditions} с {social_package} и с {benefit}, {schedule}',
            '{professional_development}, {employment_type} и {work_environment},
{schedule}',
                        'Предоставление {equipment} и {additional_benefits},
{work_life_balance}',
            '{work_life_balance} с {professional_development}',
            '{schedule} в {work_environment} с {benefit}, {work_life_balance}',
                  '{professional_development}, {employment_type} с {benefit} и
{additional_benefits}',
                    '{employment_type} {social_package} и {work_life_balance},
{additional_benefits}',
            '{benefit}, {schedule}, {social_package}',
                              '{work_conditions} + {additional_benefits} +
{professional_development}',
                  '{employment_type} с полным пакетом: {benefit}, {equipment},
{work_life_balance}',
                        'Комплексные условия: {work_conditions}, {schedule},
{work_environment}',
                  'Социальный пакет: {social_package} и {additional_benefits},
{employment_type}, {work_life_balance}',
                          'Развитие и баланс: {professional_development} и
{work_life_balance}'
        ]
```

```
        # 25
        benefits = [
            'Обустроенное рабочее место', 'ДМС', 'Официальное трудоустройство',
              'Рабочий автомобиль от компании', 'Служебное жилье', 'Компенсация
питания',
               'Спортивный зал', 'Корпоративный детский сад', 'Оплата мобильной
связи', 'Транспортные расходы',
                'Компенсация проезда', 'Оплата парковки', 'Бесплатное питание',
'Фрукты и снеки в офисе',
              'Кофе и напитки', 'Дресс-код', 'Корпоративная библиотека', 'Игровая
комната',
                'Места для отдыха', 'Душевые кабины', 'Зарядка для гаджетов',
'Wi-Fi',
                'Служебный транспорт', 'Такси за счет компании', 'Велосипедная
парковка'
        ]

        # 20
        work_conditions = [
               'Работа в большой компании', 'Работа в международной компании',
'Стабильная компания',
                    'Удаленная работа', 'Гибкий график работы', 'Возможность
командировок',
               'Современный офис в центре города', 'Динамичная рабочая среда',
'Открытая планировка офиса',
                     'Эргономичная мебель', 'Кондиционирование', 'Отопление',
'Видеонаблюдение',
                   'Пропускной режим', 'Парковка для сотрудников', 'Зоны для
коворкинга',
              'Переговорные комнаты', 'Тихие кабинеты', 'Техническая поддержка',
'IT инфраструктура'
        ]

        # 18
        schedules = [
               'График работы 2 на 2', 'Гибкий рабочий график', 'Полный рабочий
день',
                'Неполный рабочий день', 'Сменный график', 'Пятидневная рабочая
неделя',
                'Вахтовый метод работы', 'Удаленный график работы', 'Свободный
график',
                 'Посменная работа', 'Ночные смены', 'Выходные по скользящему
графику',
```

```
                'Сокращенная неделя', 'Гибкое начало дня', 'Компенсирующие
выходные',
            'Отпуск в удобное время', 'Возможность сверхурочных', 'Выходные по
требованию'
        ]

        # 22
        equipment = [
               'ноутбука', 'корпоративного телефона', 'планшета', 'оргтехники',
'средств защиты',
                'специализированного оборудования', 'служебного автомобиля',
'форменной одежды',
            'рабочей униформы', 'защитных касок', 'перчаток', 'очков защиты',
                 'мобильного интернета', 'виртуального номера', 'программного
обеспечения',
                'лицензионных программ', 'облачного хранилища', 'виртуального
рабочего места',
               'удаленного доступа', 'VPN подключения', 'корпоративной почты',
'мессенджеров'
        ]

        # 20
        employment_types = [
               'Официальное трудоустройство', 'Трудовой договор', 'Контрактная
система',
            'Аутсорсинг', 'Фриланс', 'Частичная занятость', 'Проектная работа',
               'Стажировка с последующим трудоустройством', 'Сезонная работа',
'Временный контракт',
                'Бессрочный договор', 'Срочный договор', 'Самозанятость', 'ИП
сотрудничество',
             'Агентский договор', 'Лизинг персонала', 'Аутстаффинг', 'Подрядные
работы',
            'Волонтерство', 'Пробный период'
        ]

        # 25
        work_environments = [
             'Работа в дружном коллективе', 'Молодая команда', 'Профессиональное
окружение',
            'Международная команда', 'Стартап среда', 'Корпоративная культура',
             'Инновационная атмосфера', 'Креативное пространство', 'Дружелюбная
атмосфера',
```

```
                'Поддерживающее окружение', 'Мотивирующая среда', 'Конкурентная
обстановка',
             'Сплоченная команда', 'Разнообразный коллектив', 'Мультикультурная
среда',
                'Инклюзивное пространство', 'Безбарьерная среда', 'Экологичный
офис',
            'Зеленые зоны', 'Природное освещение', 'Комфортный микроклимат',
             'Пространство для collaboration', 'Зоны для brainstorming', 'Тихое
рабочее место',
            'Открытое коммуникационное пространство'
        ]

        # 28
        social_packages = [
               'Полный социальный пакет', 'Медицинская страховка', 'Страхование
жизни',
                  'Пенсионные накопления', 'Отпуск 30 дней', 'Дополнительные
выходные',
                'Оплачиваемый больничный', 'Материальная помощь', 'Декретный
отпуск',
                'Отпуск по уходу за ребенком', 'Учебный отпуск', 'Творческий
отпуск',
                      'Компенсация лекарств', 'Стоматологическая страховка',
'Оздоровительные программы',
                   'Санитарно-курортное лечение', 'Профилактические осмотры',
'Вакцинация',
                'Медицинские чекапы', 'Психологическая помощь', 'Юридическая
поддержка',
             'Нотариальные услуги', 'Финансовое консультирование', 'Налоговое
консультирование',
                'Ипотечные программы', 'Кредитные каникулы', 'Подарки детям',
'Новогодние подарки'
        ]

        # 30
        additional_benefits = [
            'Премии по результатам работы', 'Бонусы за перевыполнение плана',
              'Опционы на акции компании', 'Корпоративные скидки', 'Обучение за
счет компании',
                'Карьерный рост', 'Участие в конференциях', 'Языковые курсы',
'Годовые бонусы',
                'Квартальные премии', 'Ежемесячные KPI', 'Процент от продаж',
'Проектные бонусы',
```

```
            'Реферальные программы', 'Бонусы за рекомендации', 'Надбавки за
стаж',
            'Надбавки за квалификацию', 'Компенсация обучения', 'Стипендии',
            'Гранты на образование', 'Участие в хакатонах', 'IT сертификации',
            'Профессиональные стандарты', 'Патентные вознаграждения', 'Авторские
отчисления',
              'Лицензионные платежи', 'Роялти', 'Дивиденды', 'Доля в компании',
'Stock options'
        ]

        # 25
        work_life_balance = [
                'Возможность удаленной работы', 'Гибкое начало рабочего дня',
'Компенсационный отдых',
               'Корпоративный отдых', 'Семейные мероприятия', 'Психологическая
поддержка', 'Фитнес-программы',
                'Йога в офисе', 'Медитационные комнаты', 'Массажные кресла',
'Настольные игры',
             'Корпоративные спортивные мероприятия', 'Турниры по киберспорту',
'Тимбилдинги',
                  'Выездные мероприятия', 'Ретриты', 'Воркшопы по балансу',
'Тайм-менеджмент тренинги',
              'Коучинг по балансу', 'Детские комнаты', 'Няни на мероприятиях',
'Семейные дни',
                'Возможность работы из дома', 'Гибкий график для родителей',
'Поддержка родителей'
        ]

        # 28
        professional_development = [
            'Обучение и тренинги', 'Карьерные консультации', 'Наставничество',
             'Профессиональные сертификации', 'Участие в воркшопах', 'Стажировки
за рубежом',
              'Курсы повышения квалификации', 'MBA программы', 'Мастер-классы от
экспертов',
                'Внутренние обучающие программы', 'Внешние тренинги', 'Онлайн
курсы',
              'Профессиональная литература', 'Технические воркшопы', 'Soft skills
тренинги',
            'Лидерские программы', 'Менторские программы', 'Коучинг сессии',
             'Партнерские программы обучения', 'Академические отпуска', 'Научная
деятельность',
```

```
                'Публикации в журналах', 'Участие в исследованиях', 'Патенты и
изобретения',
                'Профессиональные конкурсы', 'Олимпиады и чемпионаты', 'Спикерские
программы',
            'Экспертные сообщества'
        ]

        template = self.random_element(templates)

        result = template.format(
            benefit=self.random_element(benefits),
            work_conditions=self.random_element(work_conditions),
            schedule=self.random_element(schedules),
            equipment=self.random_element(equipment),
            employment_type=self.random_element(employment_types),
            work_environment=self.random_element(work_environments),
            social_package=self.random_element(social_packages),
            additional_benefits=self.random_element(additional_benefits),
            work_life_balance=self.random_element(work_life_balance),

professional_development=self.random_element(professional_development)
        )

        return result


class PostProvider(BaseProvider):
    def post_name(self):
        templates = [
            '{position}',
            '{seniority} {position}',
            '{specialization} {position}',
            '{position} {department}',
            '{seniority} {specialization} {position}',
            '{position} {direction} направления',
            '{scope} {position}',

            '{seniority} {position} {department}',
            '{specialization} {position} {direction}',
            '{position} {department} {scope}',
            '{seniority} {specialization} {position} {department}',
            '{position} {direction} ({scope})',
            '{seniority} {position} по {specialization}',
```

```
            '{position} с фокусом на {focus}',
            '{seniority} {position} в {department}',
            '{position} ({technology} направление)',
            '{specialization} {position} {level} уровня',
            '{position} - {product} команда',
            '{seniority} {position} для {market}',

            '{seniority} {specialization} {position} {department} {direction}',
            '{position} {scope} с экспертизой в {technology}',
            '{seniority} {position} ({specialization}) {department}',
            '{position} {level} уровня в {department}'
        ]

        # 35
        positions = [
            'Разработчик', 'Программист', 'Инженер', 'Аналитик', 'Архитектор',
                      'Менеджер', 'Дизайнер', 'Тестировщик', 'Администратор',
'Консультант',
            'Специалист', 'Эксперт', 'Координатор', 'Руководитель', 'Директор',
               'Стартап-лид', 'Тимлид', 'Проджект-менеджер', 'Продакт-менеджер',
'Скрам-мастер',
              'DevOps-инженер', 'Data Scientist', 'ML-инженер', 'AI-специалист',
'Бизнес-аналитик',
                'Системный аналитик', 'Технический писатель', 'UX/UI дизайнер',
'Графический дизайнер',
              'Веб-разработчик', 'Мобильный разработчик', 'Фулстек разработчик',
'Бэкенд разработчик',
            'Фронтенд разработчик', 'QA инженер'
        ]

        # 15
        seniorities = [
            'Младший', 'Начинающий', 'Стажер', 'Junior', 'Средний',
            'Опытный', 'Старший', 'Ведущий', 'Senior', 'Главный',
            'Principal', 'Штатный', 'Ключевой', 'Эксперт', 'Team Lead'
        ]

        # 25
        specializations = [
                    'программного обеспечения', 'баз данных', 'веб-приложений',
'мобильных приложений',
```

```
                    'искусственного интеллекта', 'машинного обучения',
'кибербезопасности', 'облачных технологий',
                'больших данных', 'блокчейн', 'IoT', 'компьютерного зрения',
'естественного языка',
            'робототехники', 'геймдев', 'финансовых технологий', 'медицинских
технологий',
                'образовательных технологий', 'телекоммуникационных систем',
'сетевых технологий',
              'встраиваемых систем', 'микросервисной архитектуры', 'DevOps',
'Agile', 'Scrum'
        ]

        # 30
        departments = [
            'отдела разработки', 'IT отдела', 'технического отдела', 'отдела
аналитики',
                'отдела дизайна', 'отдела тестирования', 'отдела качества',
'проектного офиса',
            'продуктовой команды', 'исследовательской группы', 'инновационной
лаборатории',
                'дата-центра', 'облачной платформы', 'мобильной разработки',
'веб-разработки',
                'бэкенд разработки', 'фронтенд разработки', 'полного цикла',
'клиентской части',
                    'серверной части', 'инфраструктуры', 'безопасности',
'производительности',
                'пользовательского опыта', 'интерфейсов', 'бизнес-логики',
'интеграций',
            'миграции данных', 'технической поддержки'
        ]

        # 20
        directions = [
            'технического', 'продуктового', 'бизнес', 'исследовательского',
            'инновационного', 'стратегического', 'операционного', 'клиентского',
            'партнерского', 'международного', 'цифрового', 'технологического',
            'аналитического', 'развития', 'автоматизации', 'оптимизации',
            'трансформации', 'внедрения', 'поддержки', 'обслуживания'
        ]

        # 18
        scopes = [
            'Коммерческий', 'Корпоративный', 'Стартап', 'Продуктовый',
```

```
                                'Проектный',  'Технический',  'Функциональный',
'Кросс-функциональный',
            'Междисциплинарный', 'Международный', 'Локальный', 'Федеральный',
            'Региональный', 'Глобальный', 'Нишевый', 'Массовый', 'B2B', 'B2C'
        ]

        # 28
        focuses = [
                    'масштабируемость систем', 'производительность приложений',
'безопасность данных',
                'пользовательский опыт', 'автоматизацию процессов', 'аналитику
данных',
            'машинное обучение', 'компьютерное зрение', 'обработку естественного
языка',
            'микросервисную архитектуру', 'контейнеризацию', 'облачные решения',
                    'мобильную разработку', 'веб-технологии', 'API разработку',
'интеграцию систем',
                'миграцию в облако', 'DevOps практики', 'Agile методологии',
'тестирование качества',
            'мониторинг систем', 'логирование ошибок', 'резервное копирование',
                    'катастрофоустойчивость', 'производительность баз данных',
'кэширование данных',
            'оптимизацию запросов', 'индексацию данных'
        ]

        # 22
        technologies = [
               'Python', 'Java', 'JavaScript', 'TypeScript', 'C++', 'C#', 'Go',
'Rust',
            'Kotlin', 'Swift', 'PHP', 'Ruby', 'Scala', 'SQL', 'NoSQL', 'React',
            'Angular', 'Vue', 'Node.js', 'Docker', 'Kubernetes', 'AWS'
        ]

        # 15
        levels = [
            'начального', 'базового', 'среднего', 'продвинутого', 'экспертного',
            'профессионального', 'ведущего', 'старшего', 'младшего', 'главного',
                        'принципального',  'архитектурного',  'стратегического',
'тактического', 'операционного'
        ]

        # 20
        products = [
```

```
            'корпоративных решений', 'мобильных приложений', 'веб-платформ',
            'облачных сервисов', 'AI продуктов', 'аналитических систем',
            'финтех решений', 'медицинских систем', 'образовательных платформ',
            'игровых продуктов', 'социальных сетей', 'маркетплейсов',
            'CRM систем', 'ERP систем', 'SCM систем', 'BI решений',
            'IoT платформ', 'блокчейн решений', 'VR/AR приложений', 'чат-ботов'
        ]

        # 18
        markets = [
                    'корпоративного сектора', 'малого бизнеса', 'стартапов',
'государственных организаций',
            'образования', 'здравоохранения', 'финансов', 'ритейла', 'телекома',
              'медиа и развлечений', 'логистики', 'недвижимости', 'автомобильной
промышленности',
                'энергетики', 'сельского хозяйства', 'строительства', 'туризма',
'игровой индустрии'
        ]

        template = self.random_element(templates)

        result = template.format(
            position=self.random_element(positions),
            seniority=self.random_element(seniorities),
            specialization=self.random_element(specializations),
            department=self.random_element(departments),
            direction=self.random_element(directions),
            scope=self.random_element(scopes),
            focus=self.random_element(focuses),
            technology=self.random_element(technologies),
            level=self.random_element(levels),
            product=self.random_element(products),
            market=self.random_element(markets)
        )

        return result
```

Название файла: generate.py

```
from faker import Faker
from datetime import datetime, timedelta
import random
```

```
from create import *
from providers import *

fake = Faker('ru_RU')
fake.add_provider(SkillRequirementProvider)
fake.add_provider(EducationNameProvider)
fake.add_provider(FieldOfActivityProvider)
fake.add_provider(ConditionProvider)
fake.add_provider(PostProvider)

def generate_passports(count):
    passports = []
    passport_numbers = set()
    for i in range(count):
        while True:
            passport_number = [
                fake.random_int(min=1000, max=9999),
                fake.random_int(min=100000, max=999999)
            ]
            str_passport = str(passport_number[0]) + str(passport_number[1])
            if str_passport not in passport_numbers:
                passport_numbers.add(str_passport)

                passports.append(Passport(
                    passport_number=passport_number[0],
                    passport_series=passport_number[1]
                ))
                break
    return passports


def generate_tins(count):
    tins = []
    tin_numbers = set()
    for i in range(count):
        while True:
            tin = fake.random_int(min=100000000000, max=999999999999)
            if tin not in tin_numbers:
                tin_numbers.add(tin)

                tins.append(Tin(
                    tin=tin
                ))
```

```
                break
    return tins


def generate_workbooks(count):
    workbooks = []
    workbook_numbers = set()
    for i in range(count):
        while True:
            workbook_number = fake.random_int(min=1000000, max=9999999)
            if workbook_number not in workbook_numbers:
                workbook_numbers.add(workbook_number)

                workbooks.append(Workbook(
                    workbook_number=workbook_number,
                                issue_date=fake.date_between(start_date='-30y',
end_date='today')
                ))
                break
    return workbooks

def generate_workers(count, passports, tins):
    workers = []
    for i in range(count):
        worker = Worker(
            passport_id = passports[i].passport_id,
            tin_id = tins[i].tin_id,
            birthday = fake.date_of_birth(minimum_age=18, maximum_age=80)
        )
        workers.append(worker)
    return workers


def generate_worker_workbooks(count, workers, workbooks):
    worker_workbooks = []
    for i in range(count):

        worker_workbooks.append(WorkerWorkbook(
            worker_id = workers[i].worker_id,
            workbook_id = workbooks[i].workbook_id
        ))
    return worker_workbooks
```

```
def generate_educations(count):
    educations = []
    names = set()
    for i in range(count):
        while True:
            name = fake.education_name()
            if name not in names:
                names.add(name)
                if "Колледж" in name:
                    type = EducationType.secondary
                else:
                    type = EducationType.higher

                educations.append(Education(
                    name = name,
                    type = type
                ))
                break
    return educations


def generate_worker_educations(workers, educations):
    worker_educations = []
    for worker in workers:
        count_educations = random.randint(0, 3)
        selected_educations = random.sample(educations, count_educations)

        for education in selected_educations:
            enroll_date = fake.date_between(
                start_date = worker.birthday + timedelta(days=365*14),
                end_date = 'today'
            )

            worker_educations.append(WorkerEducation(
                worker_id = worker.worker_id,
                education_id = education.education_id,
                enroll_date = enroll_date
            ))
    return worker_educations


def generate_posts(count=100):
```

```
    new_posts = []
    names = set()
    for i in range(count):
        while True:
            name = fake.post_name()
            if name not in names:
                names.add(name)

                new_posts.append(Post(
                    name=name
                ))
                break
    return new_posts


def generate_resumes(workers, posts):
    resumes = []
    for worker in workers:
        count_resumes = random.randint(0, 3)
        for i in range(count_resumes):

            resumes.append(Resume(
                worker_id = worker.worker_id,
                post_id = fake.random_element(posts).post_id,
                salary = fake.pyfloat(
                    left_digits=6,
                    right_digits=2,
                    min_value=40000,
                    positive=True
                )
            ))
    return resumes


def generate_skills(count):
    skills = []
    descriptions = set()
    for i in range(count):
        while True:
            description = fake.skill_requirement_description()
            if description not in descriptions:
                descriptions.add(description)
```

```
                skills.append(Skill(
                    description = description
                ))
                break
    return skills


def generate_resume_skills(resumes, skills):
    resume_skills = []
    for resume in resumes:

        count_skills = random.randint(0, 2)
        selected_skills = random.sample(skills, count_skills)
        for skill in selected_skills:

            resume_skills.append(ResumeSkill(
                resume_id = resume.resume_id,
                skill_id = skill.skill_id
            ))
    return resume_skills


def generate_field_of_activities(count):
    field_of_activities = []
    names = set()
    for i in range(count):
        while True:
            name = fake.field_of_activity()
            if name not in names:
                names.add(name)

                field_of_activities.append(FieldOfActivity(
                    name = name
                ))
                break
    return field_of_activities


def generate_employers(count, field_of_activities):
    employers = []
    for i in range(count):
        employers.append(Employer(
            name = fake.company(),
```

```
            activity_id = fake.random_element(field_of_activities).activity_id,
            photo_url = fake.image_url(),
            description = fake.text(max_nb_chars=200)
        ))
    return employers


def generate_presentation_file(count, employers):
    presentation_files = []
    for i in range(count):
        presentation_files.append(PresentationFile(
            name = f"Презентация {i}",
            url = fake.url(),
            employer_id = fake.random_element(employers).employer_id
        ))
    return presentation_files


def generate_workbook_writings(count):
    workbook_writings = []
    for i in range(count):
        workbook_writings.append(WorkbookWriting(
            type = fake.random_element(list(WritingType)),
            date = fake.date_between(start_date='-30y', end_date='today'),
            description = 'Молодец' if random.random () > 0.9 else None
        ))
    return workbook_writings


def generate_workbook_employer_writings(workbook_writings, workbooks,
employers):
    workbook_employer_writings = []
    for writing in workbook_writings:
        workbook_employer_writings.append(WorkbookEmployerWriting(
            writing_id = writing.writing_id,
            workbook_id = fake.random_element(workbooks).workbook_id,
            employer_id = fake.random_element(employers).employer_id,
        ))
    return workbook_employer_writings


def generate_vacancies(count, posts, employers):
    vacancies = []
```

```
    for i in range(count):
        vacancies.append(Vacancy(
            post_id = fake.random_element(posts).post_id,
            employer_id = fake.random_element(employers).employer_id,
            name = fake.job(),
            address = fake.address(),
            salary = fake.pyfloat(
                left_digits=6,
                right_digits=2,
                min_value=40000,
                positive=True
            ),
              experience = random.randint(0, 10) if random.random () > 0.5 else
None,
             comment = fake.text(max_nb_chars=200) if random.random () > 0.5 else
None,
            status = fake.random_element(list(VacancyStatus))
        ))
    return vacancies


def generate_requirements(count):
    requirements = []
    descriptions = set()
    for i in range(count):
        while True:
            description = fake.skill_requirement_description()
            if description not in descriptions:
                descriptions.add(description)

                requirements.append(Requirement(
                    description = description
                ))
                break
    return requirements


def generate_vacancy_requirements(requirements, vacancies):
    vacancy_requirements = []
    for vacancy in vacancies:
        count_requirements = random.randint(0, 3)
        selected_requirements = random.sample(requirements, count_requirements)
```

```
        for requirement in selected_requirements:
            vacancy_requirements.append(VacancyRequirement(
                vacancy_id = vacancy.vacancy_id,
                requirement_id = requirement.requirement_id
            ))
    return vacancy_requirements


def generate_conditions(count):
    conditions = []
    descriptions = set()
    for i in range(count):
        while True:
            description = fake.condition_description()
            if description not in descriptions:
                descriptions.add(description)

                conditions.append(Condition(
                    description = description
                ))
                break
    return conditions


def generate_vacancy_conditions(conditions, vacancies):
    vacancy_conditions = []
    for vacancy in vacancies:
        count_conditions = random.randint(0, 3)
        selected_conditions = random.sample(conditions, count_conditions)
        for condition in selected_conditions:
            vacancy_conditions.append(VacancyCondition(
                vacancy_id = vacancy.vacancy_id,
                condition_id = condition.condition_id
            ))
    return vacancy_conditions
```

Название файла: query.py

```
from sqlalchemy import func, and_, desc, or_, text
from create import *
import pandas as pd
from tabulate import tabulate
```

```
import time

def execute_query(db, query_number):
    if query_number == 1:
        out_data = select_vacancy_company(db)
    elif query_number == 2:
        out_data = select_vacancy_name(db)
    elif query_number == 3:
        out_data = select_count_vacancy(db)
    elif query_number == 4:
        out_data = select_archive_vacancy(db)
    elif query_number == 5:
        out_data = select_mean_salary(db)
    elif query_number == 6:
        out_data = select_worker_education(db)
    elif query_number == 7:
        out_data = select_worker_place(db)
    elif query_number == 8:
        out_data = select_vacancy_salary_experience(db)
    else:
        print("Wrong query number")
        return
     print(tabulate(out_data, headers='keys', tablefmt='plain', showindex=False,
stralign='left', numalign='left'))

def select_all(db, table_name):
    name_to_table = {
        'passport': Passport,
        'tin': Tin,
        'workbook': Workbook,
        'worker': Worker,
        'worker_workbook': WorkerWorkbook,
        'education': Education,
        'worker_education': WorkerEducation,
        'post': Post,
        'resume': Resume,
        'skill': Skill,
        'resume_skill': ResumeSkill,
        'field_of_activity': FieldOfActivity,
        'employer': Employer,
        'presentation_file': PresentationFile,
        'workbook_writing': WorkbookWriting,
        'workbook_employer_writing': WorkbookEmployerWriting,
```

```
        'vacancy': Vacancy,
        'requirement': Requirement,
        'vacancy_requirement': VacancyRequirement,
        'condition': Condition,
        'vacancy_condition': VacancyCondition
    }

    try:
        model_class = name_to_table[table_name]

        query = db.query(
            model_class
        ).all()

        print(table_name)
        for row in query:
            print(row)
    except Exception as e:
        print(f'Table {table_name} does not exist')


# 1. Какие вакансии есть у данной компании?
def select_vacancy_company(db):
    start = time.time()

    query = db.query(
        Vacancy.name.label('vacancy_name'),
        Employer.name.label('company'),
        Post.name.label('post'),
        Vacancy.salary,
        Vacancy.address,
        Vacancy.experience,
        Vacancy.comment,
        Vacancy.status
    ).join(Employer)\
    .join(Post)\
    .filter(Employer.name == 'КРКА Фарма (KRKA)') \
    .order_by('company') \
    .limit(3)\
    .all()

    end = time.time()
    print(end - start)
```

```
    out_data = pd.DataFrame([{
        'Компания': q.vacancy_name,
        'Название_вакансии': q.vacancy_name,
        'Должность': q.post,
        'Зарплата': q.salary,
        'Адрес': q.address,
        'Опыт': q.experience if q.experience else None,
        'Комментарий': q.comment if q.comment else None,
        'Статус': q.status
    } for q in query])

    return(out_data)


# 2. Какая вакансия подходит мне по названию?
def select_vacancy_name(db):
    start = time.time()

    query = db.query(
        Vacancy.name.label('vacancy_name'),
        Employer.name.label('company'),
        Post.name.label('post'),
        Vacancy.salary,
    ).join(Employer)\
    .join(Post)\
    .filter(Vacancy.name.like('%Художник%')) \
    .order_by('company') \
    .limit(3) \
    .all()

    end = time.time()
    print(end - start)

    out_data = pd.DataFrame([{
        'Название_вакансии': q.vacancy_name,
        'Компания': q.company,
        'Должность': q.post,
        'Зарплата': q.salary
    } for q in query])

    return(out_data)
```

```
# 3. Сколько поблизости вакансий от меня (указание улицы)?
def select_count_vacancy(db):
    start = time.time()

    query = db.query(
        func.count(Vacancy.name).label('vacancy_count')
    ).filter(Vacancy.address.like('%Пожар%')) \
    .order_by('vacancy_count') \
    .limit(3) \
    .all()

    end = time.time()
    print(end - start)

    out_data = pd.DataFrame([{
        'Количество_вакансий': q.vacancy_count
    } for q in query])

    return(out_data)


# 4. Какие вакансии были помещены в архив у компании?
def select_archive_vacancy(db):
    start = time.time()

    query = db.query(
        Vacancy.name.label('vacancy_name'),
        Employer.name.label('company'),
        Post.name.label('post'),
        Vacancy.salary
    ).join(Employer)\
    .join(Post)\
    .filter(and_(
        Employer.name == 'ЗАО «Кузнецова»',
        Vacancy.status == VacancyStatus.archive
    )) \
    .order_by('company') \
    .limit(3) \
    .all()

    end = time.time()
    print(end - start)
```

```
    out_data = pd.DataFrame([{
        'Название_вакансии': q.vacancy_name,
        'Компания': q.company,
        'Должность': q.post,
        'Зарплата': q.salary
    } for q in query])

    return(out_data)


# 5. Средняя ЗП каждого работодателя?
def select_mean_salary(db):
    start = time.time()

    query = db.query(
        Employer.name.label('company'),
        func.round(func.avg(Vacancy.salary), 2).label('mean_salary')
    ).join(Employer)\
    .group_by(Employer.name)\
    .order_by(desc('mean_salary')) \
    .limit(3) \
    .all()

    end = time.time()
    print(end - start)

    out_data = pd.DataFrame([{
        'Компания': q.company,
        'Средняя_зп': q.mean_salary
    } for q in query])

    return(out_data)


#  6. Сколько работников ищут работу, имея высшее образование?
def select_worker_education(db):
    start = time.time()

    query = db.query(
        func.count(func.distinct(Resume.worker_id)).label('count_worker')
    ).join(Worker)\
    .join(WorkerEducation)\
```

```
    .join(Education)\
    .filter(Education.type == EducationType.higher) \
    .order_by('count_worker') \
    .all()

    end = time.time()
    print(end - start)

    out_data = pd.DataFrame([{
        'Количество_работников': q.count_worker
    } for q in query])

    return(out_data)


# 7. Сколько работников имело более 3-х мест работы?
def select_worker_place(db):
    start = time.time()

    query = db.query(
        func.count(func.distinct(Worker.worker_id)).label('count_worker')
    ).join(WorkerWorkbook)\
    .join(Workbook)\
    .join(WorkbookEmployerWriting)\
    .join(WorkbookWriting)\
    .group_by(Worker.worker_id)\
     .having(func.count(func.distinct(WorkbookEmployerWriting.employer_id)) > 3)
\
    .order_by('count_worker') \
    .all()

    end = time.time()
    print(end - start)

    count = len(query)

    out_data = pd.DataFrame([{
        'Количество_работников': count
    }])

    return(out_data)
```

```
# 8. Какие вакансии имеют ЗП более 100 000р и не требуют опыта работы?
def select_vacancy_salary_experience(db):
    start = time.time()

    query = db.query(
        Vacancy.name.label('vacancy_name'),
        Employer.name.label('company'),
        Post.name.label('post'),
        Vacancy.salary,
        Vacancy.experience
    ).join(Employer)\
    .join(Post)\
    .filter(and_(
        Vacancy.salary > 100000,
        or_(
            Vacancy.experience == 0,
            Vacancy.experience.is_(None)
        )
    )) \
    .order_by('company') \
    .limit(3) \
    .all()

    end=time.time()
    print(end-start)

    out_data = pd.DataFrame([{
        'Название_вакансии': q.vacancy_name,
        'Компания': q.company,
        'Должность': q.post,
        'Зарплата': q.salary,
        'Опыт работы': q.experience
    } for q in query])

    return(out_data)
```