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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Базы данных»

Тема: Реализация базы данных с использованием ORM

Студент гр. 3384 _____ Рудаков А.Л.

Преподаватель _____ Михайлова С.В.

Санкт-Петербург

2025

Цель работы.

Изучить процесс создания реляционной базы данных и способы управления созданной базой данных с использованием ORM. Создать базу данных по ранее созданной модели, используя SQLAlchemy в языке Python.

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?

В данной лабораторной работе рекомендуется использовать Sequelize (Node.js). В случае, если не знаете как с ним работать, то можно воспользоваться примером из репозитория.

Вы можете использовать другой ORM по вашему выбору по согласованию с преподавателем, принимающим у вас практики.

Необходимо развернуть Sequelize на своем ПК и выполнить следующие задачи:

- Описать в виде моделей Sequelize таблицы из 1-й лабораторной работы
- Написать скрипт заполнения тестовыми данными: 5-10 строк на каждую таблицу, обязательно наличие связи между ними, данные приближены к реальности.
- Написать запросы к БД, отвечающие на вопросы из 1-й лабораторной работы с использованием ORM. Вывести результаты в консоль (или иной человеко-читабельный вывод)
- Запушить в репозиторий исходный код проекта, соблюсти .gitignore, убрать исходную базу из проекта (или иные нагенерированные данные бд если они есть).

  Описать процесс запуска: команды, зависимости
- В отчете описать цель, текст задания в соответствии с вариантом, выбранную ORM, инструкцию по запуску, скриншоты (код) моделей ORM, скриншоты на каждый запрос (или группу запросов) на изменение/таблицы с выводом результатов (ответ), ссылку на PR в приложении, вывод

Выполнение работы.

Для работы была выбрана ORM SQLAlchemy на языке python. Для работы с ней были установлены зависимости:

```
python3 -m venv venv
source ./.venv/bin/activate
pip install sqlalchemy psycopg2-binary datetime pandas tabulate
```

После установки зависимостей были написаны python-скрипты:

- create.py — хранит в себе описание таблиц через python-классы.
- insert.py — хранит в себе данные для заполнения таблиц, а также функцию, реализующую заполнение
- query.py — хранит в себе функции, выполняющие запросы к базе данных.
- main.py — хранит в себе основные функции для работы (функции вызова создания и удаления таблиц, парсер флагов в main, подключение к базе данных).

Исходный код см. в приложении А.

В программе имеются такие флаги, как:

- *–delete* - используется для полной очистки базы данных (удаляются все таблицы).
- *–create* - используется для создания таблиц базы данных
- *–fill* - используется для заполнения базы данных заранее заданными значениями.
- *–select_all*, после которого ожидается str значение с названием таблицы - используется для вывода всех записей таблицы с заданным названием.
- *–query* или *-q*, после которого ожидается int значение от 1 до 8, вызывающее запрос на выборку под заданным номером (вопросы из первой лабораторной работы).

Для создания и заполнения таблиц было выполнено:

```
python3 main.py –create –fill
```

Так как в базе данных суммарно находиться 21 таблица, приводить в отчете скриншоты всех заполненных таблиц будет неуместно, поэтому, на рис.1-2 будут приведены примеры заполненных таблиц.

```
(.venv) sun@aleksundr:~/Документы/DataBase/pythonProject$ python3 main.py --select_all passport
passport
Passport (id: 1, number: 3317, series: 627219)
Passport (id: 2, number: 3214, series: 745323)
Passport (id: 3, number: 3421, series: 231421)
Passport (id: 4, number: 3142, series: 892156)
Passport (id: 5, number: 3345, series: 324561)
Passport (id: 6, number: 3214, series: 234741)
```

Рисунок 1 - Заполненная таблица passport

```
(.venv) sun@aleksundr:~/Документы/DataBase/pythonProject$ python3 main.py --select_all workbook_writing
workbook_writing
WorkbookWriting (id: 1, type: WritingType.applying, date: 2010-01-01, description: None)
WorkbookWriting (id: 2, type: WritingType.change_position, date: 2018-02-13, description: Повышение должности)
WorkbookWriting (id: 3, type: WritingType.applying, date: 1995-07-01, description: None)
WorkbookWriting (id: 4, type: WritingType.dismissal, date: 1995-11-18, description: None)
WorkbookWriting (id: 5, type: WritingType.applying, date: 1995-12-01, description: None)
WorkbookWriting (id: 6, type: WritingType.dismissal, date: 1997-05-10, description: None)
WorkbookWriting (id: 7, type: WritingType.applying, date: 1998-01-13, description: None)
WorkbookWriting (id: 8, type: WritingType.dismissal, date: 2003-03-08, description: None)
WorkbookWriting (id: 9, type: WritingType.applying, date: 2004-05-21, description: None)
WorkbookWriting (id: 10, type: WritingType.applying, date: 2023-07-18, description: None)
```

Рисунок 2 - Заполненная таблица workbook_writing

Результат выполнения запроса “1. Какие вакансии есть у данной компании?” представлен на рис.3.

```
(.venv) sun@aleksundr:~/Документы/DataBase/pythonProject$ python3 main.py --query 1
Компания                    Название_вакансии              Должность       Зарплата    Адрес
Опыт    Комментарий                                         Статус
DataScientist для Предсказаний  DataScientist для Предсказаний  Data-Scientist  180000      Варшавская улица, 63к1, Санкт-Петербург
5       Требуется DataScientist для предсказаний кибер будущего  VacancyStatus.active
Менеджер по отбору кадров (HR)  Менеджер по отбору кадров (HR)  HR-менеджер     91000       Варшавская улица, 44, Санкт-Петербург
7       Нужен менеджер для подбора персонала в новую команду     VacancyStatus.active
```

Рисунок 3 - Результат запроса 1

Результат выполнения запроса “2. Какая вакансия подходит мне по названию?” представлен на рис.4.

```
(.venv) sun@aleksundr:~/Документы/DataBase/pythonProject$ python3 main.py --query 2
Название_вакансии          Компания    Должность      Зарплата
Начальник склада           ОкноДом     Складовщик        87000
Складовщик на новый склад  ОкноДом     Складовщик        63000
```

Рисунок 4 - Результат запроса 2

Результат выполнения запроса “3. Сколько поблизости вакансий от меня (указание улицы)?” представлен на рис.5.

```
(.venv) sun@aleksundr:~/Документы/DataBase/pythonProject$ python3 main.py --query 3
Количество_вакансий
3
```

Рисунок 5 - Результат запроса 3

Результат выполнения запроса “4. Какие вакансии были помещены в архив у компании?” представлен на рис.6.

```
(.venv) sun@aleksundr:~/Документы/DataBase/pythonProject$ python3 main.py --query 4
Название_вакансии           Компания     Должность          Зарплата
Водитель рейсового автобуса  ОАО "Везем"  Водитель автобуса  83250
```

Рисунок 6 - Результат запроса 4

Результат выполнения запроса “5. Средняя ЗП каждого работодателя?” представлен на рис.7.

```
(.venv) sun@aleksundr:~/Документы/DataBase/pythonProject$ python3 main.py -q 5
Компания           Средняя_зп
КиберПредсказания  135500
ЯнлексПитье        110000
ОАО "Везем"        83250
ОкноДом            75000
ООО "Купи Слона"   69001
```

Рисунок 7 - Результат запроса 5

Результат выполнения запроса “6. Сколько работников ищут работу, имея высшее образование?” представлен на рис.8.

```
(.venv) sun@aleksundr:~/Документы/DataBase/pythonProject$ python3 main.py -q 6
Количество_работников
5
```

Рисунок 8 - Результат запроса 6

Результат выполнения запроса “7. Сколько работников имело более 3-х мест работы?” представлен на рис.9.

```
(.venv) sun@aleksundr:~/Документы/DataBase/pythonProject$ python3 main.py -q 7
Количество_работников
1
```

Рисунок 9 - Результат запроса 7

Результат выполнения запроса “8. Какие вакансии имеют ЗП более 100 000р и не требуют опыта работы?” представлен на рис.10.

```
(.venv) sun@aleksundr:~/Документы/DataBase/pythonProject$ python3 main.py -q 8
Название_вакансии      Компания     Должность    Зарплата    Опыт работы
Курьер доставки питья  ЯнлексПитье  Курьер       110000
```

Рисунок 10 - Результат запроса 8

Результаты совпадают с результатами, полученными при написании SQL запросов в предыдущей лабораторной работе

Ссылку на Pull-Request см. в приложении Б.

Выводы.

Был изучен процесс создания реляционной базы данных и способы управления созданной базой данных с использованием ORM SQLAlchemy. С помощью SQLAlchemy в языке Python была создана база данных по ранее спроектированной модели. База данных была заполнена тестовыми данными, а также к были выполнены запросы на выборку по заданным вопросам.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Файл 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})'
```

Файл insert.py:

```
from create import *
from datetime import date

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(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()
```

Файл query.py:

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

```
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):
    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 == 'КиберПредсказания')\
    .all()

    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):
    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('%склад%'))\
    .all()

    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):
    query = db.query(
        func.count(Vacancy.name).label('vacancy_count')
    ).filter(Vacancy.address.like('%Варшавская%'))\
    .all()

    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):
    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 == 'ОАО "Везем"',
        Vacancy.status == VacancyStatus.archive
    ))\
    .all()

    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):
    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'))\
    .all()

    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):
    query = db.query(
        func.count(func.distinct(Resume.worker_id)).label('count_worker')
    ).join(Worker)\
    .join(WorkerEducation)\
    .join(Education)\
    .filter(Education.type == EducationType.higher)\
    .all()

    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):
```

```
    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)\
    .all()

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

    return(out_data)


# 8. Какие вакансии имеют ЗП более 100 000р и не требуют опыта работы?
def select_vacancy_salary_experience(db):
    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)
        )
    ))\
    .all()

    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)
```

Файл main.py:

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from create import Base
from insert import fill_database
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('--delete', action='store_true')
    parser.add_argument('--create', action='store_true')
    parser.add_argument('--fill', action='store_true')
    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(db)
        if args.select_all:
            select_all(db, args.select_all)
        if args.query:
            execute_query(db, args.query)

        db.close()

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

ПРИЛОЖЕНИЕ Б
ССЫЛКИ

Ссылка на Pull-Request: https://github.com/moevm/sql-2025-3384/pull/32