Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _ИНФОРМАТИКА, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА _____СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ_____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Парсинг данных из HeadHunter
и загрузка в СУБД PostgreSQL

Студент __ИУ5-35М_____ _____ __**И.Р. Ваксина**_____
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель _____ _____**Ю.Е. Гапанюк**_____
(Подпись, дата) (И.О.Фамилия)

2023 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ___ИУ5____
(Индекс)
_____ ___В.И. Терехов__
(И.О.Фамилия)
«_04_» ____сентября_____ 2023 г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме _____
_____

Студент группы _ИУ5-35М__

_____Ваксина Ия Романовна_____
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)
_____ИССЛЕДОВАТЕЛЬСКАЯ_____
Источник тематики (кафедра, предприятие, НИР) _____КАФЕДРА_____

График выполнения НИР: 25% к __._ нед., 50% к _._ нед., 75% к _._ нед., 100% к __._ нед.

Техническое задание *реализовать выгрузку вакансий с сайта HeadHunter, обработать данные и загрузить их в СУБД PostgreSQL.*
_____
_____
_____
_____

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на _22_ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____
_____

Дата выдачи задания «_04_» ____сентября_____ 2023 г.

Руководитель НИР _____ _____Ю.Е. Гапанюк_____
(Подпись, дата) (И.О.Фамилия)
Студент _____ _____И.Р. Ваксина_____
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Оглавление

Введение

HeadHunter (hh.ru) – это крупнейший в России и странах СНГ онлайн-ресурс для поиска работы и подбора персонала.

HeadHunter предоставляет возможность работодателям размещать вакансии и находить подходящих кандидатов, а соискателям - искать работу и откликаться на интересующие их вакансии. На сайте hh.ru зарегистрированы тысячи работодателей и миллионы соискателей.

Работодатели могут создавать профили компаний, размещать вакансии, проводить поиск кандидатов по различным критериям, а также использовать дополнительные сервисы, такие как проверка резюме на соответствие требованиям вакансии и автоматическая фильтрация кандидатов.

Соискатели могут создавать свои профили, загружать резюме, искать вакансии по различным критериям (например, по региону, зарплате, сфере деятельности), откликаться на интересующие их вакансии, а также использовать дополнительные сервисы, такие как автоматическое уведомление о новых вакансиях по заданным критериям и проверка резюме на соответствие требованиям вакансии.

Целью данной работы была выгрузка данных по вакансиям из HeadHunter в СУБД PostgreSQL для дальнейшего анализа полученной информации с помощью запросов к базе данных. Выгрузка данных была выполнена с использованием HeadHunter API – инструментария для интеграции данных с HeadHunter в проект.

1 Выгрузка вакансий с HeadHunter

HeadHunter API [1] — это инструментарий для интеграции данных с HeadHunter в свой проект.

При обращении к API для парсинга необходимо передавать параметры в get-запрос. В работе использованы следующие параметры:

- page
- per_page
- text
- area

Параметр page (number) обозначает номер страницы с вакансиями, его значение по умолчанию равно 0. С помощью per_page (number) можно задать количество элементов на странице, его значение по умолчанию равно 10 и не может быть больше 100.

При указании параметров пагинации (page, per_page) работает ограничение: глубина возвращаемых результатов не может быть больше 2000.

Значение параметра text (string) используется при поиске вакансий по полю названия вакансии. Поле вакансии для поиска можно изменить с помощью параметра search field, но в данной работе это не потребовалось.

Значение area (string) задает регион для поиска вакансий, выбираемый по id из справочника hh.ru /areas.

Данные выгружаются в файлы формата JSON. JSON-файл содержит информацию о вакансиях в виде массива, состоящего из объектов JSON. Одна вакансия представлена в виде одного объекта, хранящего данные в виде пар ключ-значение. Таким образом структуру получаемого JSON-файла можно представить следующим образом: [{“id”: “010”, “premium”: false, …}, {“id”: “011”, “premium”: false, …}, …].

В объекте JSON представлена следующая информация о вакансии: идентификационный номер, название специальности, минимальная и максимальная зарплата на должности, местоположение работы, требуемый

опыт, актуальность вакансии (не занесена ли в архив), предлагаемый график и формат работы, информацию о работодателе, раздел, к которому относится вакансия, флажок премиум-вакансии и др.

Пример объекта JSON, содержащего информацию об одной вакансии:


```
{
    "id": "7760476",
    "premium": true,
    "has_test": true,
    "response_url": null,
    "address": null,
    "alternate_url": "https://hh.ru/vacancy/7760476",
    "apply_alternate_url":
"https://hh.ru/applicant/vacancy_response?vacancyId=7760476",
    "department": {
        "id": "HH-1455-TECH",
        "name": "HeadHunter::Технический департамент"
    },
    "salary": {
        "to": null,
        "from": 100000,
        "currency": "RUR",
        "gross": true
    },
    "name": "Специалист по автоматизации тестирования (Java, Selenium)",
    "insider_interview": {
        "id": "12345",
        "url": "https://hh.ru/interview/12345?employerId=777"
    },
    "area": {
        "url": "https://api.hh.ru/areas/1",
        "id": "1",
        "name": "Москва"
    },
    "url": "https://api.hh.ru/vacancies/7760476",
    "published_at": "2013-10-11T13:27:16+0400",
    "relations": [],
    "employer": {
        "url": "https://api.hh.ru/employers/1455",
        "alternate_url": "https://hh.ru/employer/1455",
        "logo_urls": {
            "90": "https://hh.ru/employer-logo/289027.png",
```

```
            "240": "https://hh.ru/employer-logo/289169.png",
            "original": "https://hh.ru/file/2352807.png"
        },
        "name": "HeadHunter",
        "id": "1455"
    },
    "response_letter_required": false,
    "type": {
        "id": "open",
        "name": "Открытая"
    },
    "archived": "false",
    "working_days": [
        {
            "id": "only_saturday_and_sunday",
            "name": "Работа только по сб и вс"
        }
    ],
    "working_time_intervals": [
        {
            "id": "from_four_to_six_hours_in_a_day",
            "name": "Можно работать сменами по 4-6 часов в день"
        }
    ],
    "working_time_modes": [
        {
            "id": "start_after_sixteen",
            "name": "Можно начинать работать после 16-00"
        }
    ],
    "accept_temporary": false,
    "experience": {
      "id": "noExperience",
      "name": "Нет опыта"
    },
    "employment": {
      "id": "full",
      "name": "Полная занятость"
    }
}
```


Выгрузка данных с сайта будет производиться по алгоритму представленному на рисунке 1.

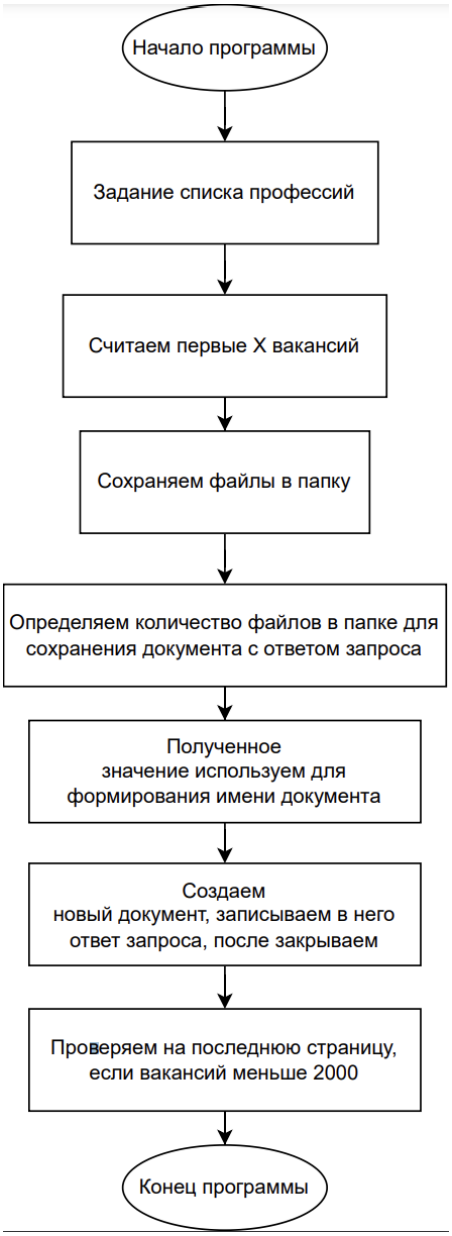

Рисунок 1 – Алгоритм парсинга данных

Код программы:

```
import requests
import json
import time
import os


def getPages(page=0, t=''):
    params = {
        'text': t,
        'area': 1,
        'page': page,
        'per_page': 100}

    req = requests.get('https://api.hh.ru/vacancies', params)
    data = req.content.decode()
    req.close()
    return data
```

```
txt = ['Кладовщик', 'Менеджер', 'Помощник', 'Администратор', 'Секретарь',
'Делопроизводитель', 'Актер', 'Тестировщик', 'Уборщик', 'Няня', 'Сборщик',
'Оператор', 'Укладчик', 'Строитель', 'Директор', 'Монтажник', 'Техник',
'Консультант', 'Бармен', 'Флорист', 'Бизнес-аналитик', 'Программист',
'Разработчик', 'Аналитик', 'Логист', 'Дизайнер', 'Курьер', 'Водитель',
'Продавец', 'Инженер', 'Кассир', 'Автослесарь', 'Слесарь', 'Сантехник',
'Риэлтор', 'Технолог', 'Повар', 'Бригадир', 'Брокер', 'Бухгалтер', 'Переводчик',
'Аниматор', 'Ведущий специалист', 'Ветеринар', 'Воспитатель', 'Врач', 'Учитель',
'Косметолог', 'Массажист', 'Парикмахер', 'Главный бухгалтер', 'Мастер маникюра',
'Рабочий', 'Промоутер', 'Партнер', 'Педагог', 'Пекарь', 'Ассистент', 'Печатник',
'Тренер', 'Подсобный рабочий', 'Портной', 'Практикант', 'Преподаватель',
'Приемщик', 'Проектировщик', 'Производитель работ', 'Прораб', 'Психолог',
'Медсестра', 'Дворник', 'Дежурный', 'Диспетчер', 'Дистрибьютор', 'Журналист',
'Заведующая магазином', 'Заведующий библиотекой', 'Заведующий кассой',
'Заместитель бухгалтера', 'Заместитель начальника', 'Заместитель руководителя',
'Заместитель управляющего', 'Инженер-проектировщик', 'Инженер ПТО',
'Инструктор', 'Инструктор', 'Кадровый специалист', 'Клиентский менеджер',
'Клинер', 'Комплектовщик', 'Кондитер', 'Конструктор', 'Контролер',
'Координатор', 'Копирайтер', 'Лаборант', 'Маркетолог', 'Маркировщик', 'Мастер',
'Машинист', 'Математик', 'Медбрат', 'Медицинский представитель', 'Мерчендайзер',
'Методист', 'Методолог', 'Механик', 'Модель', 'Наладчик', 'Налоговый',
'Начальник', 'Начинающий', 'Грузчик', 'Онлайн', 'Операционист', 'Операционный
управляющий', 'Охранник', 'Оценщик', 'Сервисный', 'Системный администратор',
'Системный аналитик', 'Сметчик', 'Сотрудник']
# Считаем первые X вакансий
for t in txt:
    for page in range(0, 20):
        jsObj = json.loads(getPages(page, t))

        # Сохраняем файлы в папку
        # Определяем количество файлов в папке для сохранения документа с
ответом запроса
        # Полученное значение используем для формирования имени документа
        nextFileName =
'C:/DZ_PDB/vacancies/{}.json'.format(len(os.listdir('C:/DZ_PDB/vacancies')))

        # Создаем новый документ, записываем в него ответ запроса, после
закрываем
        f = open(nextFileName, mode='w', encoding='utf8')
        f.write(json.dumps(jsObj, ensure_ascii=False, indent=3))

        # Проверка на последнюю страницу, если вакансий меньше 2000
        if (jsObj['pages'] - page) <= 1:
            break

        time.sleep(0.25)

print('Страницы поиска собраны!')
```

Так как на сайте есть ограничения на глубину парсинга (2000 вакансий), то необходимо запустить несколько циклов парсинга с различными словами (профессиями) для фильтрации поиска, чтобы минимизировать выбор повторяющихся вакансий. Переменная txt представляет собой массив слов, по которым реализуется цикл для поиска вакансий. Данные слава передаются в запросе в качестве параметра text. Также для минимизации повторов вакансий

данная программа была запущена 5 раз для поиска вакансий в разны городах (Москва, Санкт-Петербург, Новосибирск, Краснодар, Екатеринбург).

Парсинг 2, 16 Гб данных занял примерно 5 часов.

2 Выгрузка данных в PostgreSQL

Далее выполним загрузку данных в БД PostgreSQL [2]. Создадим три связные таблицы «vacancies» (основные данные по вакансии), «employer» (список компаний) и «metro» (список станций метро). Алгоритм загрузки данных в БД представлен на рисунке 2.

Рисунок 2 – Алгоритм загрузки данных в БД

Код программы:

```
from sqlalchemy import create_engine
import pandas as pd
import json
import os
```

```
# Создание соединения с БД
engine = create_engine("postgresql://postgres:123456@127.0.0.1:5432/myDb1")
conn = engine.connect()

ids = []
premium = []
name = []
department_id = []
department_name = []
has_test = []
response_letter_required = []
area_id = []
area_name = []
area_url = []
salary_from = []
salary_to = []
salary_currency = []
salary_gross = []
type_id = []
type_name = []
address_city = []
address_street = []
address_building = []
address_lat = []
address_lng = []
address_description = []
address_raw = []
address_metro_station_name = []
address_metro_line_name = []
address_metro_station_id = []
address_metro_line_id = []
address_metro_lat = []
address_metro_lng = []
address_id = []
published_at = []
created_at = []
archived = []
apply_alternate_url = []
url = []
alternate_url = []
employer_id = []
employer_name = []
employer_url = []
employer_alternate_url = []
employer_logo_urls_90 = []
employer_logo_urls_240 = []
employer_logo_urls_original = []
employer_vacancies_url = []
employer_trusted = []
snippet_requirement = []
snippet_responsibility = []
accept_temporary = []
professional_roles_id = []
professional_roles_name = []
accept_incomplete_resumes = []

s = 0

listDir = os.listdir('vacancies1')
twoHalfList = [listDir[:(len(listDir) // 2)], listDir[(len(listDir) // 2):]]

for halfList in twoHalfList:
    for fileName in halfList:
        if (s % 100 == 0):
```

```
        print(s)
    s += 1

    # Чтение данных из JSON файла
    file = open(f'vacancies1/{fileName}', encoding='utf8')
    jsonText = json.loads(file.read())
    file.close()

    # Создание списков для записи данных
    for item in jsonText['items']:
        ids.append(item['id'])
        premium.append(item['premium'])
        name.append(item['name'])

        try:
            department_id.append(item['department']['id'])
            department_name.append(item['department']['name'])

        except TypeError:
            department_id.append(None)
            department_name.append(None)

        has_test.append(item['has_test'])
        response_letter_required.append(item['response_letter_required'])
        area_id.append(item['area']['id'])
        area_name.append(item['area']['name'])
        area_url.append(item['area']['url'])

        try:
            salary_from.append(item['salary']['from'])
            salary_to.append(item['salary']['to'])
            salary_currency.append(item['salary']['currency'])
            salary_gross.append(item['salary']['gross'])

        except TypeError:
            salary_from.append(None)
            salary_to.append(None)
            salary_currency.append(None)
            salary_gross.append(None)

        type_id.append(item['type']['id'])
        type_name.append(item['type']['name'])

        try:
            address_city.append(item['address']['city'])
            address_street.append(item['address']['street'])
            address_building.append(item['address']['building'])
            address_lat.append(item['address']['lat'])
            address_lng.append(item['address']['lng'])
            address_raw.append(item['address']['raw'])
            address_id.append(item['address']['id'])

        except TypeError:
            address_city.append(None)
            address_street.append(None)
            address_building.append(None)
            address_lat.append(None)
            address_lng.append(None)
            address_raw.append(None)
            address_id.append(None)

        try:
            address_description.append(item['address']['description'])
```

```
            except TypeError:
                address_description.append(None)

            try:
address_metro_station_name.append(item['address']['metro']['station_name'])
address_metro_line_name.append(item['address']['metro']['line_name'])
address_metro_station_id.append(item['address']['metro']['station_id'])
               address_metro_line_id.append(item['address']['metro']['line_id'])
                address_metro_lat.append(item['address']['metro']['lat'])
                address_metro_lng.append(item['address']['metro']['lng'])

            except TypeError:
                address_metro_station_name.append(None)
                address_metro_line_name.append(None)
                address_metro_station_id.append(None)
                address_metro_line_id.append(None)
                address_metro_lat.append(None)
                address_metro_lng.append(None)

            published_at.append(item['published_at'])
            created_at.append(item['created_at'])
            archived.append(item['archived'])
            apply_alternate_url.append(item['apply_alternate_url'])
            url.append(item['url'])
            alternate_url.append(item['alternate_url'])

            try:
                employer_id.append(item['employer']['id'])
                employer_name.append(item['employer']['name'])
                employer_url.append(item['employer']['url'])
                employer_alternate_url.append(item['employer']['alternate_url'])
                employer_vacancies_url.append(item['employer']['vacancies_url'])
                employer_trusted.append(item['employer']['trusted'])

            except KeyError:
                employer_id.append(None)
                employer_name.append(None)
                employer_url.append(None)
                employer_alternate_url.append(None)
                employer_vacancies_url.append(None)
                employer_trusted.append(None)

            try:
               employer_logo_urls_90.append(item['employer']['logo_urls']['90'])
             employer_logo_urls_240.append(item['employer']['logo_urls']['240'])
   employer_logo_urls_original.append(item['employer']['logo_urls']['original'])

            except (TypeError, KeyError):
                employer_logo_urls_90.append(None)
                employer_logo_urls_240.append(None)
                employer_logo_urls_original.append(None)

            snippet_requirement.append(item['snippet']['requirement'])
            snippet_responsibility.append(item['snippet']['responsibility'])
            accept_temporary.append(item['accept_temporary'])
            professional_roles_id.append(item['professional_roles'][0]['id'])
           professional_roles_name.append(item['professional_roles'][0]['name'])
            accept_incomplete_resumes.append(item['accept_incomplete_resumes'])

    print('Первая часть завершена')

    # Создание датафрейма, который сохраняется в БД в таблицу vacancies
    df = pd.DataFrame({'id': ids,
                       'fk_employer_id': employer_id,
```

```
                            'fk_address_metro_station_id': address_metro_station_id,
                            'premium': premium,
                            'name': name,
                            'department_id': department_id,
                            'department_name': department_name,
                            'has_test': has_test,
                            'response_letter_required': response_letter_required,
                            'area_id': area_id,
                            'area_name': area_name,
                            'area_url': area_url,
                            'salary_from': salary_from,
                            'salary_to': salary_to,
                            'salary_currency': salary_currency,
                            'salary_gross': salary_gross,
                            'type_id': type_id,
                            'type_name': type_name,
                            'address_city': address_city,
                            'address_street': address_street,
                            'address_building': address_building,
                            'address_lat': address_lat,
                            'address_lng': address_lng,
                            'address_description': address_description,
                            'address_raw': address_raw,
                            'address_id': address_id,
                            'published_at': published_at,
                            'created_at': created_at,
                            'archived': archived,
                            'apply_alternate_url': apply_alternate_url,
                            'url': url,
                            'alternate_url': alternate_url,
                            'snippet_requirement': snippet_requirement,
                            'snippet_responsibility': snippet_responsibility,
                            'accept_temporary': accept_temporary,
                            'professional_roles_id': professional_roles_id,
                            'professional_roles_name': professional_roles_name,
                            'accept_incomplete_resumes': accept_incomplete_resumes})

    df.to_sql('vacancies', con=conn, schema=None, if_exists='append',
index=False)

    # Создание датафрейма, который сохраняется в БД в таблицу employer
    u_employer_id = []
    u_employer_name = []
    u_employer_url = []
    u_employer_alternate_url = []
    u_employer_logo_urls_90 = []
    u_employer_logo_urls_240 = []
    u_employer_logo_urls_original = []
    u_employer_vacancies_url = []
    u_employer_trusted = []

    for i in range(len(employer_id)):

        if employer_id[i] not in u_employer_id:
            u_employer_id.append(employer_id[i])
            u_employer_name.append(employer_name[i])
            u_employer_url.append(employer_url[i])
            u_employer_alternate_url.append(employer_alternate_url[i])
            u_employer_logo_urls_90.append(employer_logo_urls_90[i])
            u_employer_logo_urls_240.append(employer_logo_urls_240[i])
            u_employer_logo_urls_original.append(employer_logo_urls_original[i])
            u_employer_vacancies_url.append(employer_vacancies_url[i])
            u_employer_trusted.append(employer_trusted[i])
```

```
    df = pd.DataFrame({'employer_id': u_employer_id,
                       'employer_name': u_employer_name,
                       'employer_url': u_employer_url,
                       'employer_alternate_url': u_employer_alternate_url,
                       'employer_logo_urls_90': u_employer_logo_urls_90,
                       'employer_logo_urls_240': u_employer_logo_urls_240,
                       'employer_logo_urls_original':
u_employer_logo_urls_original,
                       'employer_vacancies_url': u_employer_vacancies_url,
                       'employer_trusted': u_employer_trusted
                       })

    df.to_sql('employer', con=conn, schema=None, if_exists='append',
index=False)

    # Создание датафрейма, который сохраняется в БД в таблицу metro
    u_address_metro_station_id = []
    u_address_metro_station_name = []
    u_address_metro_line_id = []
    u_address_metro_line_name = []
    u_address_metro_lat = []
    u_address_metro_lng = []

    for i in range(len(address_metro_station_id)):

        if address_metro_station_id[i] not in u_address_metro_station_id:
            u_address_metro_station_id.append(address_metro_station_id[i])
            u_address_metro_station_name.append(address_metro_station_name[i])
            u_address_metro_line_id.append(address_metro_line_id[i])
            u_address_metro_line_name.append(address_metro_line_name[i])
            u_address_metro_lat.append(address_metro_lat[i])
            u_address_metro_lng.append(address_metro_lng[i])

    df = pd.DataFrame({'address_metro_station_id': u_address_metro_station_id,
                       'address_metro_station_name':
u_address_metro_station_name,
                       'address_metro_line_id': u_address_metro_line_id,
                       'address_metro_line_name': u_address_metro_line_name,
                       'address_metro_lat': u_address_metro_lat,
                       'address_metro_lng': u_address_metro_lng,
                       })

    df.to_sql('metro', con=conn, schema=None, if_exists='append', index=False)
    print('Вторая часть завершена')

# Закрываем соединение с БД
conn.close()
```

При парсинге данных было получено 7272 файла с данными. При загрузке данных в БД необходимо сначала извлечь данные из файлов, а потом загрузить в БД. Так как объем данных большой, произошло переполнение оперативной памяти (4Гб). При попытке извлечь данные сразу из всех файлов программа зависала из-за переполнения памяти. Чтобы решить данную проблему, данные извлекались из 500 файлов за 4 цикла (т.е. по 125 файлов за раз). Загрузка данных в четыре таблице заняла примерно 2 часа.

При загрузке данных в БД на ноутбуке с объемом оперативной памяти 8 Гб такой проблемы не возникло, и загрузка данных заняла примерно 5 минут.

Схема полученной базы данных изображена на рисунке 3.

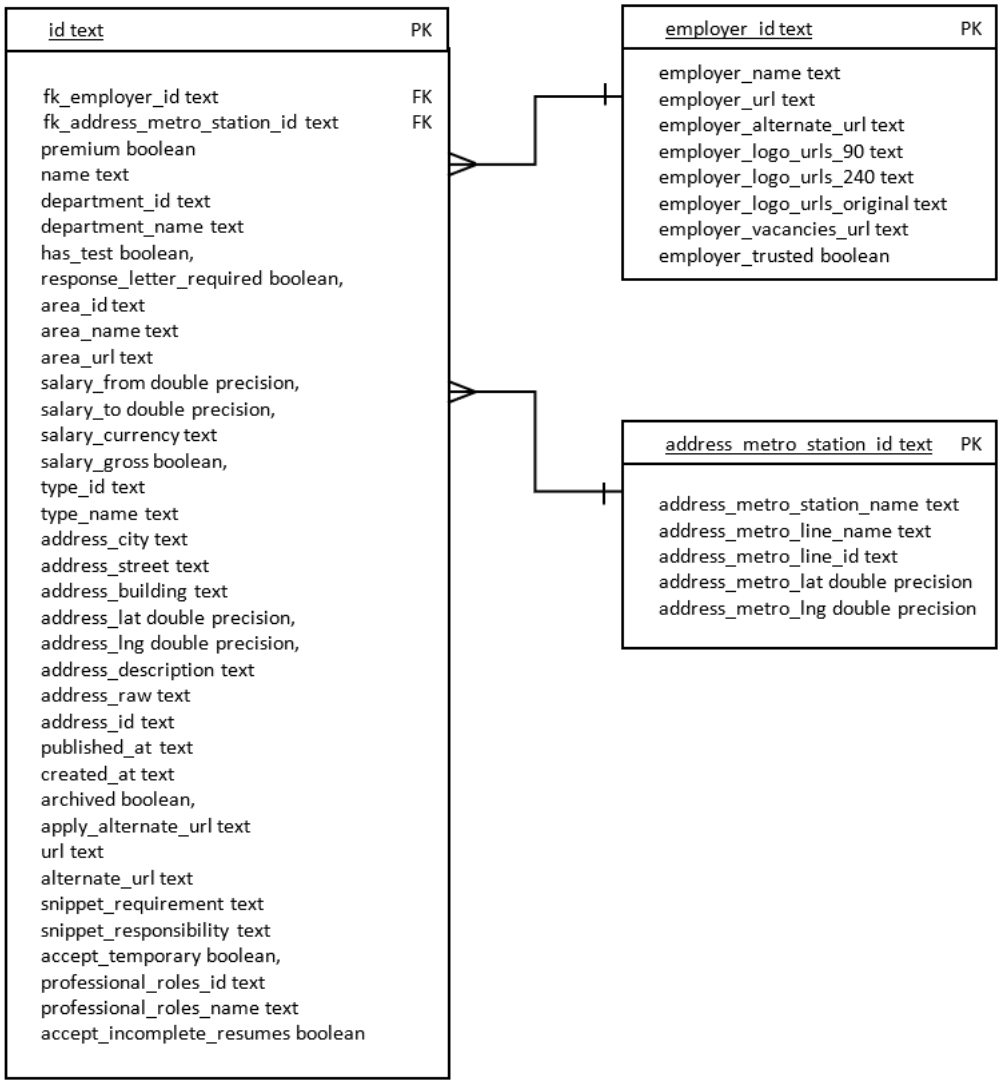

Рисунок 3 – Схема базы данных

Таблица «employer» хранит информацию о компаниях: название компании, ссылка на сайт компании, ссылки на страницы компании на сайте hh.ru, ссылка на страницу вакансий компании на сайте hh.ru.

Таблица «metro» содержит информацию о станциях метро: название станции, название линии метро, номера (id) станции и линии метро, координаты станции метро.

Таблица «vacancies» содержит информацию о вакансиях: название вакансии, название профессии, требования к кандидатам, обязанности будущего работника, адрес, размер заработной платы, статус вакансии, дата

создания объявления, дата публикации, альтернативные ссылки на объявление. Также есть поля, которые реализуют связь с таблицами «metro» и «employer» - fk_employer_id и fk_address_metro_station_id text.

Описание всех атрибутов в базы данных:

Таблица Vacancies:

- id text – идентификационный номер вакансии
- premium boolean – флаг премиум-вакансии
- department_id text – идентификационный номер раздела вакансий
- department_name text – название раздела вакансий
- has_test text – есть тестовое задание
- response_letter_required boolean – необходимость сопроводительного письма
- area_id text – идентификационный номер региона
- area_name text – название региона
- area_url text – ссылка на страницу региона
- salary_from double precision – нижняя граница зарплаты
- salary_to double precision – верхняя граница зарплаты
- salary_currency text – валюта зарплаты
- salary_gross boolean – валовая заработная плата
- type_id text – идентификационный номер типа (открытая и закрытая вакансии)
- type_name text – название типа вакансии
- address_city text – адрес: город
- address_street text – адрес: улица
- address_building text – адрес: дом
- address_lat double precision – адрес: широта
- address_lng double precision – адрес: долгота
- address_description text – описание адреса
- address_id text – идентификационный номер адреса

- published_at text – время публикации вакансии
- created_at text – время создания вакансии
- archived boolean – флаг добавления вакансии в архив
- url text – ссылка на вакансию
- alternate_url text – альтернативная ссылка на вакансию
- snippet_requirement text – требования из вакансии
- snippet_responsibility text – что попадает под ответственность кандидата на должность
- accept_temporary boolean – флаг временной работы
- professional_roles_id text – идентификационный номер профессиональной роли
- professional_roles_name text – название профессиональной роли
- accept_incomplete_resumes boolean – флаг приема неполных резюме
- fk_employer_id text – внешний ключ – идентификационный номер работодателя
- fk_address_metro_station_id text – внешний ключ – идентификационный номер метро

Таблица employer:

- employer_id text – идентификационный номер работодателя
- employer_name text – название работодателя
- employer_url text – ссылка на работодателя
- employer_alternate_url text – альтернативная ссылка на работодателя
- employer_logo_urls_90 text – ссылка на логотип работодателя
- employer_logo_urls_240 text – ссылка на логотип работодателя
- employer_logo_urls_original text – ссылка на оригинал логотипа работодателя
- employer_vacancies_url text – ссылка на страницу со всеми вакансиями работодателя
- employer_trusted boolean – флаг проверенного работодателя

Таблица metro:

- address_metro_station_id text – идентификационный номер станции метро
- address_metro_station_name text – название станции метро
- address_metro_line_name text – название линии метро
- address_metro_line_id text – идентификационный номер линии метро
- address_metro_lat double precision – широта расположения станции метро
- address_metro_lng double precision – долгота расположения станции метро

В полученной базе данных все еще могут храниться одинаковые записи, для качественного анализа их необходимо удалить. Код для удаления дубликатов:

```
DELETE FROM vacancies WHERE ctid NOT IN
(SELECT max(ctid) FROM vacancies GROUP BY vacancies.*);

DELETE FROM metro WHERE ctid NOT IN
(SELECT max(ctid) FROM metro GROUP BY metro.*);

DELETE FROM employer WHERE ctid NOT IN
(SELECT max(ctid) FROM employer GROUP BY employer.*);
```

Теперь база данных готова для использования в целях анализа. Можно тестировать свои гипотезы относительно, например, корреляции между уровнем заработной платы и местом расположения компании, профессией соискателя, его уровнем; находить наиболее востребованные навыки в своей специальности для повышения шансов на трудоустройство.

Заключение

В результате работы были выгружены данные о вакансиях по 122 специальностям в 5 городах с сервиса HeadHunter в СУБД PostgreSQL с помощью HeadHunter API. Из полученной информации были удалены дубликаты, а данные были разделены на три связные таблицы в СУБД.

Теперь полученные данные можно использовать для анализа вакансий на hh.ru в целях получения статистических данных или увеличения собственных шансов на устройство на работу.

Литература

1. Head Hanter API – Текст. Изображение: электронные // Head Hanter : [сайт]. – URL: https://dev.hh.ru/

2. Введение в PostgreSQL с Python +Psycopg2 – Текст. Изображение: электронные // PythonRu : [сайт]. – URL: https://pythonru.com/biblioteki/vvedenie-v-postgresql-s-python-psycopg2