Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации
Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский технический университет связи и информатики»
(МТУСИ)

Кафедра «Программная инженерия»
дисциплина «Рефакторинг баз данных и приложений»

Лабораторная работа № 5

«Рефакторинг приложений с целью оптимизации запросов и индексов базы данных»

Выполнил: студент группы БПИ2303
Березовец Илья Сергеевич

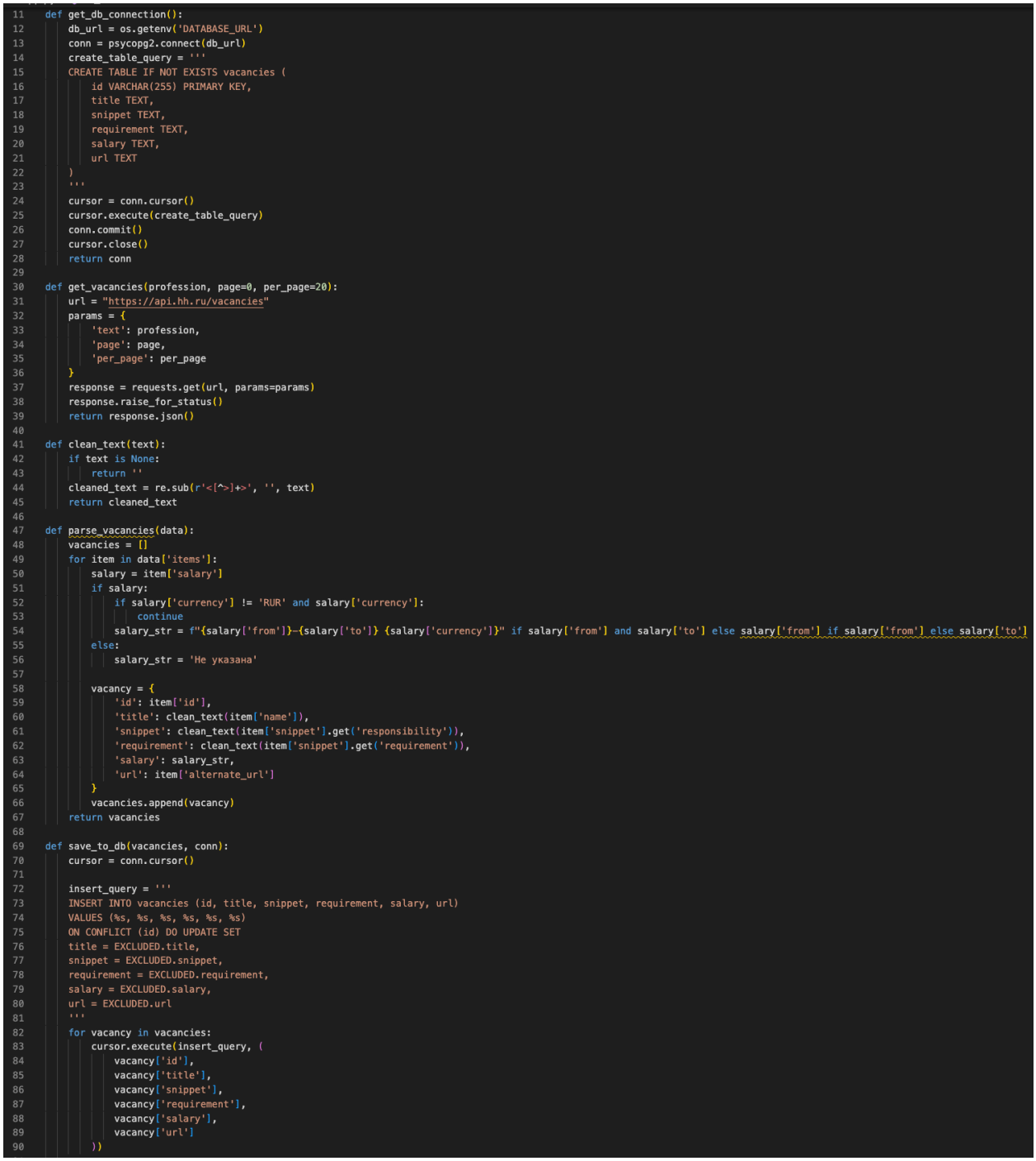
Москва 2025

Цели лабораторной работы

1. Изучить основные принципы оптимизации SQL-запросов.
2. Научиться выявлять узкие места в работе базы данных.
3. Освоить создание, модификацию и удаление индексов для ускорения запросов.
4. Применить знания на практике для оптимизации базы данных и приложения.

Ход работы:

Изначальный код:



```python
def get_db_connection():
    db_url = os.getenv('DATABASE_URL')
    conn = psycopg2.connect(db_url)
    create_table_query = '''
    CREATE TABLE IF NOT EXISTS vacancies (
        id VARCHAR(255) PRIMARY KEY,
        title TEXT,
        snippet TEXT,
        requirement TEXT,
        salary TEXT,
        url TEXT
    )
    '''
    cursor = conn.cursor()
    cursor.execute(create_table_query)
    conn.commit()
    cursor.close()
    return conn

def get_vacancies(profession, page=0, per_page=20):
    url = "https://api.hh.ru/vacancies"
    params = {
        'text': profession,
        'page': page,
        'per_page': per_page
    }
    response = requests.get(url, params=params)
    response.raise_for_status()
    return response.json()

def clean_text(text):
    if text is None:
        return ''
    cleaned_text = re.sub(r'<[^>]+>', '', text)
    return cleaned_text

def parse_vacancies(data):
    vacancies = []
    for item in data['items']:
        salary = item['salary']
        if salary:
            if salary['currency'] != 'RUR' and salary['currency']:
                continue
            salary_str = f"{salary['from']}-{salary['to']} {salary['currency']}" if salary['from'] and salary['to'] else salary['from'] if salary['from'] else salary['to']
        else:
            salary_str = 'Не указана'

        vacancy = {
            'id': item['id'],
            'title': clean_text(item['name']),
            'snippet': clean_text(item['snippet'].get('responsibility')),
            'requirement': clean_text(item['snippet'].get('requirement')),
            'salary': salary_str,
            'url': item['alternate_url']
        }
        vacancies.append(vacancy)
    return vacancies

def save_to_db(vacancies, conn):
    cursor = conn.cursor()

    insert_query = '''
    INSERT INTO vacancies (id, title, snippet, requirement, salary, url)
    VALUES (%s, %s, %s, %s, %s, %s)
    ON CONFLICT (id) DO UPDATE SET
    title = EXCLUDED.title,
    snippet = EXCLUDED.snippet,
    requirement = EXCLUDED.requirement,
    salary = EXCLUDED.salary,
    url = EXCLUDED.url
    '''
    for vacancy in vacancies:
        cursor.execute(insert_query, (
            vacancy['id'],
            vacancy['title'],
            vacancy['snippet'],
            vacancy['requirement'],
            vacancy['salary'],
            vacancy['url']
        ))
```

Рис. 1 – (первая часть изначального кода)

```
def save_to_db(vacancies, conn):
            ))

    conn.commit()
    cursor.close()

def salary_to_numeric(salary_str):
    if isinstance(salary_str, int):
        return salary_str
    if salary_str == 'Не указана':
        return 0
    try:
        parts = salary_str.split('-')
        if len(parts) == 2:
            return (int(parts[0].strip()) + int(parts[1].strip())) // 2
        return int(parts[0].strip())
    except (ValueError, IndexError):
        return 0

@app.route('/', methods=['GET', 'POST'])
def index():
    """
    Main page to search for vacancies
    ---
    parameters:
      - name: profession
        in: query
        type: string
        required: true
        description: Profession to search for
      - name: num_pages
        in: query
        type: integer
        required: false
        default: 5
        description: Number of pages to fetch
    responses:
      200:
        description: A list of vacancies
        schema:
          type: array
          items:
            type: object
            properties:
              id:
                type: string
              title:
                type: string
              snippet:
                type: string
              requirement:
                type: string
              salary:
                type: string
              url:
                type: string
    """
    vacancies = []
    profession = ''
    num_vacancies = 0

    if request.method == 'POST':
        profession = request.form['profession']
        num_pages = int(request.form.get('num_pages', 5))
        conn = get_db_connection()

        all_vacancies = []
        for page in range(num_pages):
            data = get_vacancies(profession, page=page)
            vacancies_page = parse_vacancies(data)
            all_vacancies.extend(vacancies_page)

        save_to_db(all_vacancies, conn)
        conn.close()

        vacancies = all_vacancies
        num_vacancies = len(vacancies)

    return render_template('index.html', vacancies=vacancies, profession=profession, num_vacancies=num_vacancies)
```

Рис. 2 – (вторая часть изначального кода)

```
    cursor.execute('''
    SELECT title, ROUND(AVG(
        CASE
            WHEN position('-' in salary) > 0 THEN
                (CAST(substring(salary, '(\d+)-') AS bigint) + CAST(substring(salary, '-(\d+)') AS bigint)) / 2
            ELSE CAST(REGEXP_REPLACE(salary, '[^0-9]', '', 'g') AS bigint)
        END
    )) AS average_salary
    FROM vacancies
    WHERE salary <> 'Не указана' AND salary LIKE '%RUR%'
    GROUP BY title
    ORDER BY average_salary DESC
    LIMIT 3
    ''')
    top_salaries = cursor.fetchall()

    top_salaries = [{'name': row[0], 'average_salary': int(row[1])} for row in top_salaries]

    cursor.close()
    conn.close()
    return render_template('all_vacancies.html', vacancies=vacancies, vacancy_count=vacancy_count, top_salaries=top_salaries)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Рис. 3 – (третья часть изначального кода)

Применю следующие улучшения коду, а именно SQL – запросам.

1. Индексы: добавлю индекс на title и salary — часто используются в выборке и группировке.
2. Уберу SELECT *: заменю выборки на конкретные столбцы.
3. Оптимизирую SQL-запросы: исправлю выражения с substring, чтобы избежать ненужных операций.
4. Добавлю анализ через EXPLAIN.

1.

```
# Индексы — для ускорения WHERE, GROUP BY, ORDER BY
cursor.execute('CREATE INDEX IF NOT EXISTS idx_title ON vacancies(title)')
cursor.execute('CREATE INDEX IF NOT EXISTS idx_salary ON vacancies(salary)')
```

Рис. 4 - (Добавил индексы)ё

2.

```
cursor.execute('SELECT id, title, snippet, requirement, salary, url FROM vacancies')
```

Рис. 5 – (Добавил выборку на конкретные столбцы)

3.

```
  (CAST(split_part(salary, '-', 1) AS bigint) + CAST(split_part(split_part(salary, '-', 2), ' ', 1) AS bigint)) / 2
```

Рис. 6 – (Исправил substring на split_part)

4.

```
cursor.execute('EXPLAIN ANALYZE SELECT title, salary FROM vacancies WHERE salary <> \'Не указана\' AND salary LIKE \'%RUR%\' ORDER BY title')
```

Рис. 7 – (Добавил анализ через EXPLAIN)

Резюмирую улучшения:
Вот кратко от твоего лица:

1. Добавил индексы на `title` и `salary`, так как они часто участвуют в фильтрации и группировке — это ускоряет запросы.

2. Заменил `SELECT *` на выбор конкретных столбцов — уменьшает нагрузку и повышает читаемость.
3. Оптимизировал SQL-запросы, переписал `substring`-выражения, чтобы убрать лишние операции.
4. Добавил `EXPLAIN ANALYZE` для анализа запросов и оценки их производительности.

Вывод:
1. Изучил основные принципы оптимизации SQL-запросов.
2. Научился выявлять узкие места в работе базы данных.
3. Освоил создание, модификацию и удаление индексов для ускорения запросов.
4. Применил знания на практике для оптимизации базы данных и приложения.

Ссылка на GitHub: https://github.com/iberezaa/RBDiP