1. Введение.

В docker-compose указаны 2\* контейнера:

- БД PostgreSQL

- приложение на Python, которое собирает данные

- \* закомментирован adminer на случай, если захочется поднять удобную утилиту для просмотра базы.

Приложение имеет свой Dockerfile в папке /app/src.

Для PostgreSQL прокинут порт 5432 для обращения к ней из контейнера приложения.

Запуск проверялся на следующих командах:

docker-compose build

docker-compose up

Результат работы приложения выводится в виде таблицы в консоль и в файл /app/src/results.txt.

2. Работа приложения.

2.1. Общая логика

Приложение собирает данные вакансий, обрабатывает хранящиеся вакансии и выдаёт 4 таблицы:

- 2 таблицы про все вакансии в базе;

- 2 таблицы про закрытые вакансии в базе.

После старта приложение сначала создаёт структуру БД.

Потом собирает данные из API HH.

Сначала загружаются справочники, потом вакансии.

Если вакансии нет в базе, она добавляется, если есть - её данные обновляются.

Также обновляются данные по вакансиям, которые есть в базе, но которые не попали в выборку из API.

Это потенциальные закрытые вакансии или не попавшие из-за ограничения по датам.

После окончания обработки данных, происходит сбор и вывод результатов в консоль и txt-файл.

Результаты представлены для примера, хотя есть, что покрутить, конечно.

Просто без конкретной задачи решил не углубляться.

Я выделил два поля: level и specialty.

Level - уровень (junior/middle/semior).

Specialty - специализация (frontend/backend/fullstack/mobile/ios/android/devops/data).

Заполняются они из названия вакансии.

И в результатах две таблицы:

- количество вакансий в разрезе level;

- количество вакансий в разрезе specialty.

2.2. Выборка вакансий

Выбор вакансий производится с фильтрами:

- местность (area) -- 2 (Санкт-Петербург),

- индустрия работодателя (industry) -- 7.538, 7.539, 7.540, 7.541 (это IT-индустрии),

- вид работодателя (employer) -- company (компания).

Таким образом обрабатываются вакансии IT-компании в Санкт-Петербурге.

Из-за ограничения API на количество вакансий, вакансии выбираются по следующему алгоритму:

1. Рассчитываются 5 периодов по 5 дней, начиная со дня запуска.

2. Вакансии собираются частями по рассчитанным периодам.

Далее для каждой найденной вакансии собирается подробная информация о вакансии отдельным запросом к API.

Таким образом собираются все возможные полезные данные из API.

2.3. Про обработку данных вакансий (коротко).

Для каждой таблицы базы данных разработан класс-модель.

Все классы модели наследованы от базовой модели для реализации общего поведения -- вывода значений атрибутов класса в строку вида "(знач0, знач1, ... , значN)".

Это нужно для автоматизации вставки большого числа новых записей в БД.

Также разработан класс VacanciesDataParser.

Его задача -- разложить и накопить данные каждой вакансии в соответствующие собственные структуры данных, наполненные объектами классов-моделей.

Для загрузки данных в БД разработаны универсальные классы.

Один приспособлен для обработки сущностей (entity\_collector), второй -- для отношения 1:N (relation\_collector).

Логика сбора данных такая:

1. Получить данные из API.

2. Распарсить данные с помощью VacanciesDataParser.

3. Последовательно записать данные из VacanciesDataParser с помощью соответствующих collector-ов.

3.1. При записи сначала происходит выборка ключей из соответствущей таблицы БД.

3.2. Потом ключи раскладываются по "корзинам": новые, имеющиеся, старые.

3.3. По новым ключам происходит вставка (insert), по имеющимся обновление (update), по старым ТОЛЬКО ДЛЯ ВАКАНСИЙ происходит обращение к API HH для проверки, закрыта ли вакансия.

3.3.1. Если вакансия закрыта, ставится дата закрытия "сегодняшним числом".

3.3.2. Если вакансия не закрыта, дата закрытия ставится 31.12.9999 (это дата закрытия вакансии по умолчанию).