Модель данных

API работает на graph-ql, сперва были выделены схемы данных и созданы классы для сущностей:

**ReportWoodDeal** с соответственно обертками PageReportWoodDeal, Data и RootObject для корректной десериализации;

**SearchReportWoodDealResult** с соответственно обертками SearchReportWoodDealData и SearchReportWoodDealResponse.

Сделки мы будем получать, отслеживать и сохранять в базе данных. Для отслеживания изменения сделок был использован механизм хэширования сериализованной сущности алгоритмом SHA256. Для количества сущностей, которые уже имеются на сервере, его с большим запасом.  
 Код создания таблицы для сделок в БД MSSQL выглядит так:

CREATE TABLE ReportWoodDeal (

SellerName NVARCHAR(255) NOT NULL,

SellerInn CHAR(12) NOT NULL,

BuyerName NVARCHAR(255) NOT NULL,

BuyerInn CHAR(12) NOT NULL,

WoodVolumeBuyer FLOAT NOT NULL,

WoodVolumeSeller FLOAT NOT NULL,

DealDate CHAR(10) NOT NULL,

DealNumber CHAR(28) NOT NULL,

Object\_hash CHAR(44) NOT NULL,

CONSTRAINT PK\_ReportWoodDeal PRIMARY KEY (SellerInn, BuyerInn, DealNumber)

);

Ключ составной из-за нехватки вводных данных и отсутствия представлений о предметной области

DealDate – фиксированная строка, так как это облегчает процесс десериализации при получении пустой или неверной даты, существенно не увеличивая затраты на хранение и не плодит сущности, а также их Mapping, что упрощает решение.  
 ИНН – 10-12 символов, выбираем максимум.

Объемы продаж – числа с плавающей точкой, Float для c# это double для MSSQL.  
Необходимо представлять предметную область, чтобы выбрать необходимую точность,  
тогда было бы FLOAT(n), где n – число бит, используемых для хранения мантиссы.  
 Номер сделки согласно таблице состоит только из строк, начинающихся порой с 0, фиксированной длины 28 символов.

Object\_hash – это Base64-строка, полученная после SHA256 объекта, её длина постоянна и равна 44 символам.

Проверив файл robots.txt сервера, особых ограничений там не обнаружил, поэтому поставил User-Agent один из первых попавшихся для осуществления запросов.

Архитектура приложения\*

Подготовка к работе происходит так:

1. Создается объект WoodDealsRepository, отвечающий за работу с БД и хранящий строку для подключения к ней, в БД сразу создается таблица для хранения данных.
2. Создается объект QueryExecutor – обертка над HttpClient, выполняющая запросы к API.
3. Создается объекты ParsingSettings и WoodDealsPageParser – настройки и сам парсер
4. В бесконечном цикле вызывается метод парсера DoCycle(), в котором находится вся логика работы приложения.

Работа приложения состоит из трех этапов:

1. Отправка запроса на получение сущностей к API
2. Десериализация ответа
3. Итерирование по объектам, полученным в результате и:
   1. Проверка объекта на валидность
   2. Запрос к БД для поиска объекта по ключу
   3. Если объекта нет, создаем запись в БД +1 запрос
   4. Если объект есть, сравниваем хэш. Если хэш не совпал, обновляем сущность в БД +1 запрос. Иначе пропускаем сущность, т. к. она актуальна в БД.

По завершению выполнения цикла, ожидаем задержки, если время выполнения было меньше неё, иначе сразу преступаем к следующей итерации, повторяя эти 3 пункта.

Валидация данных

Основная идея в том, чтобы обеспечить целостность данных в БД, валидация грубая, но может быть просто изменена, если станет больше известно о специфике предметной области.

В самом начале проверяется наличие значений null в полях объекта – такого быть не должно.

Для дат и ИНН следующие регулярные выражения:  


К тому же есть проверка года на реальность, диапазон значений задается в коде приложения, по умолчанию 2000 и до текущего года в системе.

Поля DealNumber, BuyerName, SellerName, object\_hash проверяются на совпадение по длинне с моделью данных в БД.

Проблемы

На этапе отладки приложения была выделена проблема наличия в исходной таблице отношения объектов с одинаковыми ключами. Отличаются они лишь формой написания названия компании: ООО или ОБЩЕСТВО С ОГРАНИЧЕННОЙ ОТВЕТСТВЕННОСТЬЮ или вообще без указания типа. При получении каждый раз объекта с тем же ключом в силу специфики выполнения решения он просто обновит в БД свои данные в соответствии с самой последней полученной версией.  
 Данной проблемы можно избежать:

* + 1. Унификацией всех названий, например, менять в нашем случае при получении от сайта на ООО
    2. Установкой правила включения по названию, например, у компании не указан тип (ООО, ИП), такую сделку тогда мы в базу не занесем.

Для упрощения решения это не было сделано, но данный функционал несложно внедрить в решение, если это будет востребовано.

Также можно добавить повторные попытки осуществить запрос, если соединение нестабильно, можно использовать Polly.

Пример коллизии:

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Тестирование:

1.5мбит/сек

500 – размер выборки -> 304 странички

50мс – задержка между запросами

Выполнение цикла заняло 7.5 минут, было обработано 151973 сущностей, 105912 добавлено в БД.

Пример запроса SELECT из БД после нескольких циклов работы



Контактные данные

Репозиторий решения на GitHub: https://github.com/Rrezakk/AV\_test

Трудозатраты: 12 часов

Telegram для связи: @r\_rezak\_k

Профиль на hh.ru: https://rostov.hh.ru/resume/3f64e9a0ff08e88fe20039ed1f775458536679