Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Проект по предметот Неструктурирани бази на податоци и XML

Екстрахирање на податоци од Etsy, нивно препроцесирање, складирање во MongoDB и манипулирање со истите

Изработиле:

Андреј Анчевски - 175006
Марио Митревски - 171210
GitHub: https://github.com/andrejanchevski/ecommerce_mongo_data

Web scraping and crawling

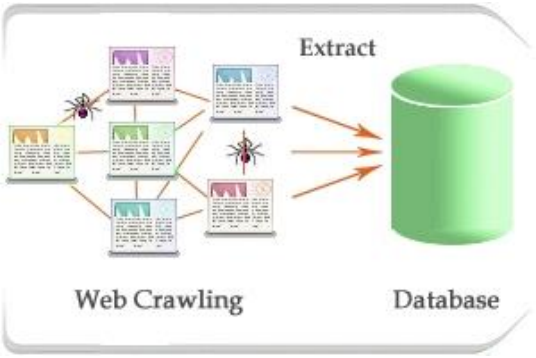
Scrapy е брза високо - функционална web scraping библиотека за crawling на веб сајтови и екстрахирање на структурирани податоци од нивните страници. Се користи во разни намени од податочно рударство, податочен мониторинг, процесирање на информации или автоматско тестирање.

Scrapy оригинално е дизајнирана за web scraping, но исто така може да се користи и за екстрахирање на податоци користејќи API (како Amazon Associates Web Services) или како општо наменски web crawler.

Data Scraping е процес на пронаоѓање податоци и scraping на истите. Процес кој влече податоци директно од страница. Тој процес на влечење податоци не значи влечење податоци исклучиво од вебот, туку податоците можат да бидат преземени од секаде каде што постојат податоци. Тоа значи дека може да го правиме процесот на scraping и од spreadsheets, мемориски уреди, односно секаде каде што постојат податоци во некаква форма. Овој процес е потребен за филтрирање и предефинирање на различни типови на сурови податоци во податоци што се информативни и корисни. Data scraping е процес кој е многу поспецифичен во податоците што ги екстрахира за разлика од data crawling. Алатките за data scraping можат изведат акции како java script акции, поднесување на форми, игнорирање на robots.

Web crawling процес кој оди подлабоко низ web. Кога имаме data crawling работиме со така наречени spiders или bots кои се движат и пребаруваат низ вебот за да најдат нешто што ги задоволува нашите барања. Spiders работат под дејство на алгоритам и следат точни инструкции од алгоритамот. Процесот на crawling се состои од следење на линкови до разни веб страници. Во тој момент всушност кролерите прават scraping. Не скенираат само низ страниците туку и собираат потребни информации и истовремено бараат линкови до други потребни страници.

Data Scraping	Data Crawling
Вклучува екстрахирање на информации од различни извори вклучувајќи и веб	Се однесува на преземање страници од web
Може да се направи во било кој опсег	Се прави во големи размери
Немаме дуплирање	Постои дуплирање на информации
Потребен е crawler и parser	Потребен е само crawling агент



Пример код за Spider кој екстрахира податоци за сопственици на продавници

```
Class EtsyShopSpider(scrapy.Spider):
    name = 'etsy_shop_owners_mongo'
    def start_requests(self):
        with open('etsy/shops_crawled.json') as filehandler:
            shops = json.loads(filehandler.read())
        urls = [s['shopOwnerLink'] for s in shops]
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)
    def parse(self, response, **kwargs):
        items = EtsyShopOwners()
        shop_owner_name = response.css('.wt-display-inline-flex::text').extract()
        shop_name = response.css('#content .wt-text-center-xs::text').extract_first()
        owner_location = response.css('.wt-nudge-b-1+ span::text').extract_first()
        owner_about_me = response.css('.wt-text-caption #wt-content-toggle-profile-about
p::text').extract_first()
        owner_following = 0
        owner_followers = 0
        owner_follow_info = response.css('#content .wt-text-link-no-underline span::text').extract()
        if len(owner_follow_info) > 2:
            owner_following = owner_follow_info[0]
            owner_followers = owner_follow_info[1]
        shopOwnerName = ''
        if len(shop_owner_name) > 0:
            shopOwnerName = str(shop_owner_name[0]).strip()

        items['shopName'] = str(shop_name).strip() if shop_name else ''
        items['shopOwnerName'] = shopOwnerName
        items['ownerFollowing'] = owner_following
        items['ownerFollowers'] = owner_followers
        items['ownerLocation'] = owner_location if owner_location else ''
        items['ownerAboutMe'] = owner_about_me if owner_about_me else ''

        yield items
```

По извршување на командата scrapy crawl etsy_shop_owners_mongo, всушност тоа е име на горе прикажаниот Spider, Scrapy бара дефиниција на Spider и го извршува низ неговиот crawler engine.

Crawl процесот започнува така што се прават повици кон URL-а кои се генерирани во функцијата start_requests(self), или најчесто URL-та се преставуваат како низа start_urls која што е дефинирана во класата на Spider. По секое барање се повикува callback методот parse така што се праќа одговорот од барањето како аргумент. Во parse методот се екстрахираат потребните податоци, во овој случај основните информации за еден сопственик на продавница и се прави yield на Scrapy Item со екстрахираните податоци за еден сопственик на продавница.

Една од главните предности на Scrapy е тоа што барањата се распоредуваат и се процесираат асинхроно. Тоа значи дека Scrapy не мора да чека за барање да биде завршено и обработено, туку тој во меѓувреме може да прати и друго барање или да извршува други функции. Тоа значи дека голем број на барања ќе се прават и доколку некое барање падне или пак се случи некоја грешка додека се справуваме со одредено барање. Со ова овозможуваме брз начин на crawling бидејќи праќаме повеќе истовремени барања на fault – tolerant начин. Scrapy исто така ни овозможува контрола над коректноста на процесот на crawling преку поставување download delay помеѓу секое барање, лимитирање на количината на паралелни барања по домен или по IP.

Функционалности на Scrapy:

- Вградена поддршка за селектирање и екстрахирање податоци од HTML/XML извори користејќи CSS селектори и XPath изрази, со помошни методи за екстрахирање користејќи регуларни изрази.
- Вградена поддршка за генерирање на експорт фајлови каде што може да се складираат податоците кои се scraped во соодветен формат и да се користат низ други системи. Тие експорт фајлови може да се во JSON, CSV или XML формат.
- Интерактивна shell конзола за испробување на CSS селектори и XPath изрази за користење која што е вистински погодна при дебагирање.
- Широк опсег на вградени екстензии и middleware за справување со
    - Колачиња и сесии
    - HTTP карактеристики како компресија, автентикација, кеширање
    - robots.txt
    - Рестрикции за crawling во длабочина
    - User – agent spoofing
- Telnet конзола која што се прикачува на Python конзола која работи во Scrapy процесот со цел да се направи интроспекција и да се дебагира процесот на crawling.

Елементи на Spider

Според горниот пример за Spider се забележува дека секој нов креиран Spider наследува од класата scrapy.Spider и со тоа се дефинираат некои атрибути и методи како:

- name - идентификатор за Spider. Мора да е уникатно во еден проект, не смееме во еден проект да имаме исто име за различни Spiders.
- start_reqeusts() - се генерира листа од барања од каде што Spider ќе започне со процесот на crawling. Понатамошните барања ќе се генерираат следствено по иницијалните барања.

- parse() - метод кој се повикува со цел да се справи со одговорот преземен за секое направено барање. Одговорот како објект е инстанца од TextResponse што всушност ја содржи содржината на страната за кое сме направиле барање и тој објект во себе си носи методи кои ни овозможуваат справување со добиената содржина.

Scrapy ги распоредува scrapy.Request објектите кои што се вратени од start_requests методот на Spider. По примање на одговорите за секое барање, инстанцира Response обекти и го повикува callback методот поврзан за секое барање и како аргумент на тој метод го праќа одговорот.

Алтернатива за start_requests() методот кој што генерира scrapy.Request објекти од URL-а е декларирање на start_urls класен атрибут како листа од URL-а. Таа листа потоа ќе се искористи од стандардната имплементација на start_requests() која што ќе ги креира иницијалните барања за одреден Spider.

```
class EtsyShopSpider(scrapy.Spider):
    name = 'etsy_shops'
    start_urls = [
        'https://www.etsy.com/search/shops?page=' + str(i) for i in range(1, 20)
    ]

    def parse(self, response, **kwargs):
        links = response.css('.wrap a::attr(href)').extract()
        for link in links:
            yield response.follow(str(link), callback=self.parse_shop)
```

Селектори

При процесот на crawling на веб страници најчестата задача која што се извршува е екстрахирањето на податоци од HTML изворот. Scrapy доаѓа со сопствен механизам за екстрахирање на податоци така наречени селектори кои што бираат одредени елементи од HTML документот, а додека специфицирањето на кои делови да бидат одбрани се врши со XPath или CSS изрази.

Scrapy селекторите се инстанци од Selector класата и се конструираат на тој начин што на конструкторот за Selector се предава TextResponse објект врз кој ќе се прават селекциите. Кога користиме Scrapy ни овозможува екстензија, така што не мораме да ги конструираме Scrapy селекторите рачно, бидејќи response аргументот кој го има во callback методот за парсирање во себе ги содржи методите css() и xpath().

```
from scrapy.selector import Selector
from scrapy.http import HtmlResponse
Response=HtmlResponse(url='https://www.etsy.com/search/shops?page=1',body=body)
Selector(response=response).css('.shop-icon-external::attr(src)').extract()

response.css('#listing-page-cart .wt-break-word').xpath('string()').extract()

productShopName = response.css('#listing-page-cart .wt-text-body-01 .wt-text-link-no-underline
span::text').extract()
```

Со овој selector кажуваме дека сакаме да ги селектираме сите елементи кои што имаат id со вредност listing-page-cart и потоа нивните деца кои имаат класа wt-text-body-01, потоа децата на тие елементи кои имаат класа wt-text-link-no-undeline и во тие елементи да го земеме span елементот и текстуалната вредност во него.

За да можеме да го направиме самиот момент на екстрахирање податоци мораме по селектирањето да повикаме некој од методите кои ќе го направат вистинското извлекување на податоци бидејќи всушност методите css() и xpath() враќаат SelectorList инстанци кои всушност преставуваат нови селектори.

- get - го зема првиот елемент кој го задоволува условот за селектирање
- getAll - ги зема сите елементи кои го задоволуваат условот за селектирање и ги враќа резултатите во листа
- extract - алијас на getAll()
- extract_first - алијас на get()

Доколку условите за селектирање не успеат да пронајдат никаков елемент кој го задоволува наведениот услов погоре наведените методи враќаат None.

Екстензии на CSS селектори

- ::text - селектирање на текстуални јазли
- :: attr(name) каде што name специфицира името на атрибутот на одреден елемент за кој се зима вредноста

```
shopLink = response.css('#listing-page-cart .wt-text-body-01 .wt-text-link-no-
underline::attr(href)').extract()
```

Со можноста на selector да враќа инстанца од SelectorList всушност тие резултати може да се користат како посебен response и на нив да се извршат посебни, поспецификни селекции. Ова овозможува итерирање низ повеќе елементи кои се дел од одреден SelectorList.

```
shopFeedbackSlots = response.css('.reviews-list .review-item')
for shopFeedbackSlot in shopFeedbackSlots:
       shopFeedbackItem = EtsyShopFeedbacksMongo()
           feedback_reviewer = shopFeedbackSlot.css('.flag-body .shop2-review-attribution
       a::text').extract_first()
           feedback_reviewer_link = shopFeedbackSlot.css('.flag-body .shop2-review-attribution
       a::attr(href)').extract_first()
```

XPath селектори

XPath изразите се многу моќна алатка и претставуваат основа за Scrapy Selectors. CSS селекторите во позадина всушност се претвораат во XPath изрази. Со помош на XPath изразите не само што можеме да навигираме низ структурата, туку може и да гледаме во содржината на елементите. Користејќи XPath можеме да селектираме линк кој што го содржи текстот “Next page”. Во многу случаеви каде што е потребно да се земе првото дете или директниот наследник на одредена компонента во тој случај функционалностите на XPath изразите ни нудат голема флексибилност

```
feedback_rating = shopFeedbackSlot.css('.flag-body .mb-xs-0 .stars-
svg')  .xpath("input[@name='rating']/@value").extract_first()
```

```
feedback_date = shopFeedbackSlot.css('.flag-body .shop2-review-
attribution').xpath('string()') .extract_first()
```

Регуларни изрази

Selector инстанцата овозможува метод re() кој што е метод за екстрахирање на податоци користејќи регуларни изрази. За разлика од xpath() и css() методите, re() како резултат враќа листа од стрингови.

```
response.xpath('//a[contains(@href, "image")]/text()').re(r'Name:\s*(.*)')
```

Item Exporters

Често користена потребна карактеристика при процесот на scraping е можноста да се складираат екстрахираните податоци соодветно и тоа најчесто се прави со генерирање на одреден експорт фајл.

За полесно екстрахирање можеме да користиме Item Exporters каде што ќе генерираме Items во форма каква што сакаме да ни бидат податоците кои ќе ги добиеме во експорт документите како резултат на scraping процесот. Потребно е генераторот на одредениот Spider да направи yield на иницијализарниот Item и со тоа податоците во експортираниот фајл ќе ја имаат формата дефинирана во самиот Item.

```
class EtsyProductWithFeedbacksMongo(scrapy.Item):
    productName = scrapy.Field()
    productShopName = scrapy.Field()
    shopLink = scrapy.Field()
    productNoOfReviews = scrapy.Field()
    productRating = scrapy.Field()
    productPrice = scrapy.Field()
    productOriginalPrice = scrapy.Field()
    productReviews = scrapy.Field()
    productItems = scrapy.Field()
    categories = scrapy.Field()


class EtsyProductReviewsMongo(scrapy.Item):
    feedbackRating = scrapy.Field()
    feedbackText = scrapy.Field()
    feedbackReviewer = scrapy.Field()
    feedbackDate = scrapy.Field()
```

Download Middleware преставува фрејмворк од хукови кој што влијае во процесирањето на request/response процесирање. Проблемот кој се крена при процесот на scraping е појавата на status error code 429 - “Too many requests”. Причината за појавување на тој статусен код е брзото праќање кон Etsy на голем број барања и scraping-от не продолжуваше. Со цел да го решиме проблемот поставивиме свој Download Middleware каде што секој пат кога Spider ќе најде на тој статусен код застануваше и чекаше 1 минута пред да го прати следниот повик.

```
class TooManyRequestsRetryMiddleware(RetryMiddleware):

    def __init__(self, crawler):
        super(TooManyRequestsRetryMiddleware, self).__init__(crawler.settings)
        self.crawler = crawler

    @classmethod
    def from_crawler(cls, crawler):
        return cls(crawler)

    def process_response(self, request, response, spider):
        if request.meta.get('dont_retry', False):
            return response
        elif response.status == 429:
            self.crawler.engine.pause()
            time.sleep(60) # If the rate limit is renewed in a minute, put 60 seconds, and so
on.
            self.crawler.engine.unpause()
            reason = response_status_message(response.status)
            return self._retry(request, reason, spider) or response
        elif response.status in self.retry_http_codes:
            reason = response_status_message(response.status)
            return self._retry(request, reason, spider) or response
        return response
```

Рекурзивен scraping на податоци:

```
class EtsyShopSpider(scrapy.Spider):
    name = 'etsy_categories'
    start_urls = [
        'https://www.etsy.com/help/categories/seller'
    ]

    def parse(self, response, **kwargs):
        categoryList = response.css('.category-list')
        yield from self.ul_parsing(categoryList, parentCategory="")

    def ul_parsing(self, categoryList, parentCategory):

        listElements = categoryList.xpath('./li')
        for li in listElements:
            items = EtsyCategoryItem()
            categoryName = li.xpath('./span/text()').extract()
            items['categoryName'] = categoryName
            items['parentCategoryName'] = parentCategory
            yield items
            subList = li.xpath('./ul')
            if len(subList) != 0:
                yield from self.ul_parsing(subList, categoryName)
            else:
                continue
```

Претпроцесирање на податоци

Податоците во реалноста не се чисти. Секогаш кога работиме со голем број на податоци особено податочни множества кои сами ги генерираме од разни извори многу веројатно е дека ќе се сретнеме со неконзистентност во податоците или не целосни податоци. Претпроцесирање на податоци е техника која најчесто се користи во податочно рударење и машинско учење.

- **Некомплетни податоци** - Кога имаме големи податочни множества многу веројатно е дека одредени атрибути од интерес ќе фалат во одредени торки. Решенија:
  - Игнорирај торки - техника која е поволна кога имаме големи податочни множества и просто само ги отстрануваме торките кои немаат целосни атрибути.
  - Пополни ги вредностите кои недостасуваат - техники со кои што се пополнуваат атрибутите користејќи средна вредност од постоечките атрибути.
- **Noisy податоци** - податоци кои немаат значење и не можат да се разберат од машини. Може да се генерираат поради податочни множества кои се склони на грешки, грешки при внес на податоци
  - Binning метод - овој метод работи на сортирани податоци со цел да се порамнат. Податоците се делат на сегменти со еднаква големина и со секој сегмент одделно се прават разни методи. Во еден сегмент може да се прави замена на податоци според средна вредност на атрибути, па во друг сегмент да се користат гранични вредности со цел да ги смениме невалидните податоци.
  - Регресија - податоците се порамнуваат користејќи функција на регресија.
- **Кластерирање** - Се работи преку процесот на data in cluster.
- **Неконзистентни податоци** - податоци каде што за еден атрибут може да имаме различни типови на вредности што може да доведе до двосмисленост при користење на податоците.

Трансформирање на податоци - Трансформација на податоци во соодветна форма за процес на рударење, алгоритам за машинско учење или статистичка анализа на податоците

- Нормализација на податоци
  - Скалирање на вредностите на податоците во одреден опсег
- Избор на атрибут
  - Се конструираат нови атрибути од даденото множество на атрибути
- Дискретизација
  - Замена на сирови вредности на нумерички атрибути со нивоа на интервали

Data reduction - намалување на податоците - техника која се користи кога имаме голем волумен на податоци и анализа на истите. Целта на овие техники е да ја зголеми ефикасноста на складирање и да ги намали трошоците за анализа на податоците

- Data Cube Aggregation – се прават агрегациски операции врз податоците
- Attribute Subset Selection - во предвид се земаат значајните атрибути тие кој најмногу влијаат на податоците останатите можат да се одстранат. Се користи p – value или ниво на значајност на атрибут.
- Numerosity Reduction

- Димензионално намалување - се намалува големината на податоците користејќи механизми за енкодирање. Може да биде lossy или loseless.

Начин на претпроцесирање на податоците е чистење на податоците од null вредности така што сите редици или колони кои имаат null/nan вредности се отстрануваат. Библиотеката Pandas во Python нуди метод dropna() кој што ефикасно ги отстранува редиците и колоните од податочните множества кои имаат null/nan вредности. Со самото користење на Scrapy и exception handling методите се заштитивме од појава на null/nan вредности во нашите податоци и ја отстранивме потребата да користиме механизми кои го прават тоа.

Податочното множество кое се генерираше со помош на scraping техниките содржеше големи текстови, бидејќи во податоците имаме атрибути кој преставуваат коментар за дадена продавница или коментар за даден продукт. Коментарите оставени од корисниците содржеа разни знаци, емотикони кои по процесот на scraping се трансформираа во noisy податоци. Со самото екстрахирање на податоци исто така голем број од генерираните коментари беа со непотребни празни места кои исто така се трансформираа во noisy податоци. Бидејќи коментарите се креираат од разни корисници во нив постојат голем број на несоодветни кратенки кои во понатамошен процес при анализа или обработка на податоците можат да направат проблем. Големиот број на stop words во коментарите претставува потреба од дополнителна меморија, но со нивното елиминирање го губиме значењето на коментарот нешто што е важно во нашите податоци. За најсоодветно и чисто чување на податоци кое ќе овозможи ефикасно складирање и анализа на податоците искористивме библиотека која што ќе ги отстрани овие проблеми.

Stop Words - преставуваат често употребени зборови како што се the, a, an и најчесто голем број на search engines се направени да ги игнорираат. За да не земаат дополнителен простор во базата на податоци или да одземаат големо време на обработка дефинираме листа од зборови кои што ги сметаме за stop words. Библиотеката NLTK (Natural Language Toolkit) во Python има листа од stop words складирани во 16 различни јазици.

Текст со Stop Words	Текст без Stop Words
Geeks for Geeks - A Computer Science Portal for Geeks	Geeks for Geeks, Computer Science, Portal Geeks

```
def text_cleaner(text):
    stop_words = set(stopwords.words('english'))
    newString = text.lower()
    newString = re.sub(r'\([^)]*\)', '', newString)
    newString = re.sub('"', '', newString)
    newString = ' '.join([contraction_mapping[t] if t in contraction_mapping else t for t in
newString.split(" ")])
    newString = re.sub(r"'s\b", "", newString)
    newString = re.sub("[^a-zA-Z]", " ", newString)
    tokens = [w for w in newString.split()]
    # tokens = [w for w in newString.split() if not w in stop_words]
    long_words = []
    for i in tokens:
        if len(i) >= 0:
            long_words.append(i)
    return (" ".join(long_words)).strip()
```

Монго - База на податоци

Вовед за MongoDB

MongoDB потекнува од зборот humongous што всушност значи огромно нешто, а аналогијата доаѓа дека MongoDB базата на податоци е дел од базите на податоци кои овозможуваат чување на голем број податоци и работа со истите. Решенија од науката за бази на податоци постојат голем број и тоа од типот на PostgreSQL, Oracle, MySQL. Овие бази на податоци претставуваат структурирани бази на податоци и самата структура на податоците и постоењето на добро дефинирана шема е основната разлика со MongoDB.

Генерална разлика помеѓу MongoDB и структурирана бази на податоци

Структурираните бази на податоци се решенија каде што се складираат, дистрибуираат и нормализираат податоци низ повеќе табели од дадена шема, каде што секоја табела се состои од потребни колони за опис на истата и се градат потребни релации помеѓу табелите за да се извади контекст од податоците. Постоењето на шема доведува до чување на податоците во структуриран формат. Во нашиот креиран продукт опишан погоре тие табели изгледаа на начин како на сликите подолу.

Табела за продукти

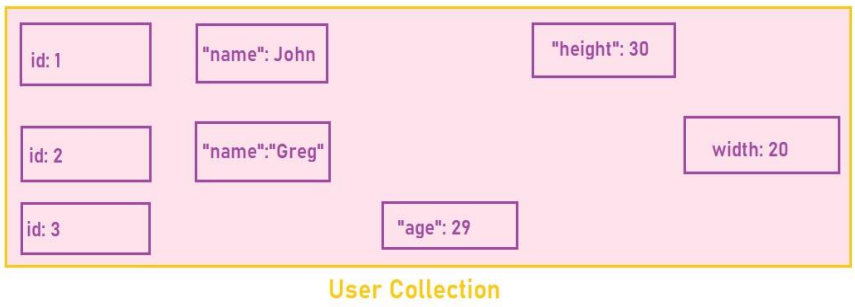
product_id	created_date	last_modified_date	deleted	price	product_description	product_name	productsku	product_category_cate
BLOB	2020-09-15 14:06:34	2020-09-15 14:06:34	0	20	This is a description for a test product	TestProduct14	111111111	1
BLOB	2020-09-15 14:06:33	2020-09-15 14:06:33	0	25	Praying emoji fun way to support our Miami Dol...	DolphinsPrayEmojiShirt	5811UTE1231	1
BLOB	2020-09-15 14:06:33	2020-09-15 14:06:33	0	30	Do you love whales? Look down before you get in	WhaleRug	564RTU123	34
BLOB	2020-09-15 14:06:33	2020-09-15 14:06:33	0	18	Tell your guests indirectly to take off their shoe...	ShoesOffRug	123SKU56U	34
BLOB	2020-09-15 14:06:34	2020-09-15 14:06:34	0	20	This is a description for a test product	TestProduct5	111111111	1
BLOB	2020-09-15 14:06:33	2020-09-15 14:06:33	0	45	If you love action heroes, this rug will definitely ...	ActionHeroRug	456UKHE12	34
BLOB	2020-09-15 14:06:32	2020-09-15 14:06:32	0	35	Enigma - Tenner Hoodie. White and clean. Your ...	ENIGMAHOODIE	123UKR123	1
BLOB	2020-09-15 14:06:34	2020-09-15 14:06:34	0	20	This is a description for a test product	TestProduct19	111111111	1
BLOB	2020-09-15 14:06:33	2020-09-15 14:06:33	0	17	Smile through emojis little chargers fans	ChargersEmojiShirt	567345UU	17
BLOB	2020-09-15 14:06:34	2020-09-15 14:06:34	0	20	This is a description for a test product	TestProduct8	111111111	1
BLOB	2020-09-15 14:06:33	2020-09-15 14:06:33	0	22	SURREAL SOMNIUM - step into the surreality	SOMNIUMSHIRT	12UHE56	1
BLOB	2020-09-15 14:06:34	2020-09-15 14:06:34	0	20	This is a description for a test product	TestProduct12	111111111	1
BLOB	2020-09-15 14:06:34	2020-09-15 14:06:34	0	20	This is a description for a test product	TestProduct4	111111111	1

Табела за продавници

shop_id	created_date	last_modified_date	shop_bank_account	shop_description	shop_logo_imag
BLOB	2020-09-15 14:06:32	2020-09-15 14:06:32	67832323111	Smile while you look down	FunnyRugsShop
BLOB	2020-09-15 14:06:32	2020-09-15 14:06:32	12309u8	You will never feel the winter with our wonderful products	CozyBlankets
BLOB	2020-09-15 14:06:32	2020-09-15 14:06:32	11111111111	This is a description for this shop	TestShop
BLOB	2020-09-15 14:06:32	2020-09-15 14:06:32	11111111111	This is a description for this shop	TestShop
BLOB	2020-09-15 14:06:32	2020-09-15 14:06:32	11111111111	This is a description for this shop	TestShop
BLOB	2020-09-15 14:06:32	2020-09-15 14:06:32	11111111111	This is a description for this shop	TestShop
BLOB	2020-09-15 14:06:32	2020-09-15 14:06:32	09809098908089	Classy wallets with your initals	EngravedLeathe
BLOB	2020-09-15 14:06:32	2020-09-15 14:06:32	11111111111	This is a description for this shop	TestShop
BLOB	2020-09-15 14:06:32	2020-09-15 14:06:32	11111111111	This is a description for this shop	TestShop
BLOB	2020-09-15 14:06:32	2020-09-15 14:06:32	0809909090344	Your head will be like on a flower garden	FlowerPillows
BLOB	2020-09-15 14:06:32	2020-09-15 14:06:32	11111111111	This is a description for this shop	TestShop
BLOB	2020-09-15 14:06:32	2020-09-15 14:06:32	11111111111	This is a description for this shop	TestShop
BLOB	2020-09-15 14:06:32	2020-09-15 14:06:32	11111111111	This is a description for this shop	TestShop

MongoDB - Основна структура

MongoDB е NoSQL решение од областа на бази на податоци каде што основната идеја е да складира податоци заедно во еден документ и не ги ограничува дизајнерите на базата на податоци на одредена шема.



Непостоењето на шема ќе не доведе до појава на така наречени messy податоци, но ни овозможува флексибилност. Флексибилноста е овозможена од следниве елементи кои се многу важен дел од MongoDB:

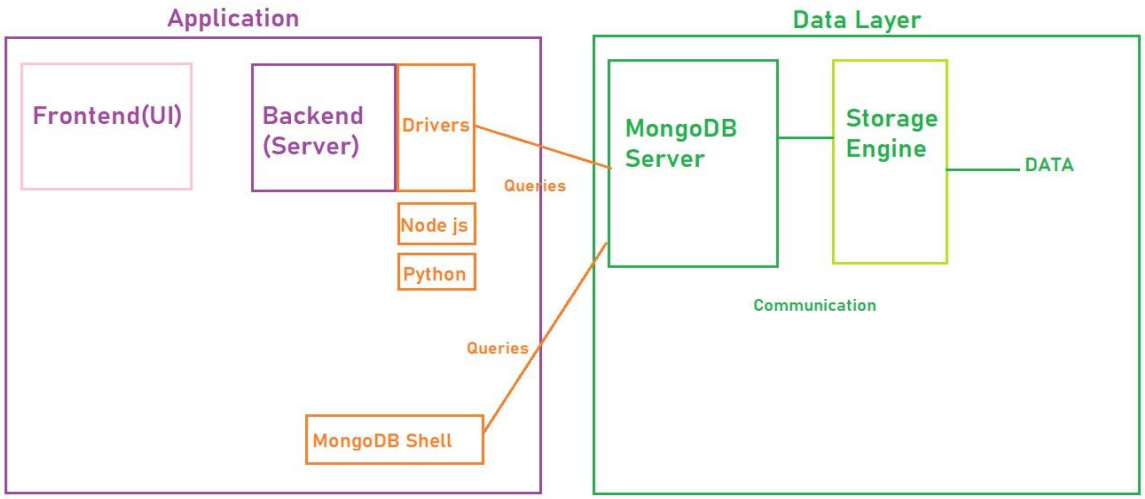
- **Колекции** (Collections)
- **Документи** (Documents) – некаде се користат и како Records (Записи)
- **JSON** (**BSON**) **формат**

Секоја база на податоци се состои од колекции, а во внатрешноста на колекциите имаме документи. Документите изгледаат на JSON објекти, но всушност во позадина тие се зачуваат како BSON формат. Флексибилноста се јавува токму во овој дел, бидејќи документите кои ги чуваме во една колекција всушност се schema less. Не следат некоја точно одредена структура како во релационите бази на податоци. Се овозможува да сместиме различен тип на податоци во иста колекција, а со тоа и можноста да расте базата на податоци како што растат потребите од апликацијата.

Во пракса постои минимално ограничување, бидејќи сепак апликација би барала одредени податоци кои и се потребни и најчесто тоа ограничување се доведува на постоење на одредени атрибути во документите низ една колекција.

Документите во JSON формат овозможуваат лесна имплементација на релациите помеѓу податоците. Документите може да се вгнездуваат едни во други, а со тоа пребарувањата низ Mongo базата на податоци стануваат поедноставни, побрзи и ефикасни. Во спротивно кај релационите бази на податоци потребно е да се прават сложени JOIN операции кои во однос на перформанси чинат многу со цел да се поврзат податоците помеѓу два ентитети. Колекција во MongoDB е аналогија на ентитет во релациона база на податоци

MongoDB има голем ефект кај апликации кои релативно често се надоградуваат. Со помош на MongoDB може лесно да се додаваат податоци од најразличен контекст во колекција кога и да е потребно. Исто така се работи и со помал број на релации. Има принцип на локализација каде што во една колекција се опфаќаат голем број на податоци. Се отстранува потребата од пристап до поголем број на табели со цел да се даде назад резултат од некое прашање. MongoDB е погодна за апликации каде што има голем проток на податоци. Пример апликација што користи голем број на сензори кои во краток временски интервал создаваат голем број на податоци. Но, исто така MongoDB ја наоѓа својата примена и во секојдневни проблематики во областа на медицината, e-commerce и многу други.



Комуникација со MongoDB
Колекции и Документи

Во MongoDB секогаш постојат една или повеќе бази на податоци. Секоја база на податоци може да се состои од една или повеќе колекции. Доколку се земе некоја аналогија колекција претставува табела во некоја од SQL базите на податоци. Колекцијата се состои од повеќе документи кои всушност преставуваат инстанци од ентитетите кои се опишани во базата. За разлика од релационите бази на податоци, каде што за креирање на шемата и табелите е потребно да се испишат DDL команди, во MongoDB базата, колекциите и документите може да се креираат on demand или имплицитно, при самото извршување на прашањето. Доколку не постојат ќе ги креира, во другиот случај ќе ги искористи веќе креираните.

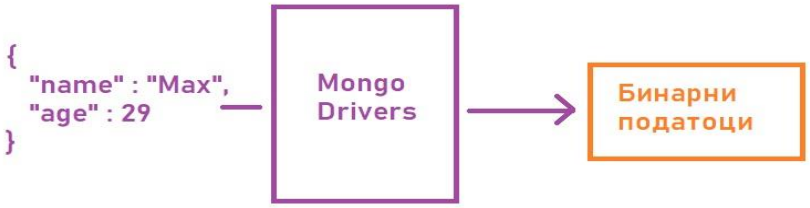
JSON & BSON

JSON документ е важен елемент од MongoDB кој се состои од клуч-вредност парови. Вредностите можат да бидат едноставни типови кои што ги познаваме како String, Number, Boolean, но може да бидат и сложени типови како други објекти или пак низи од елементи.

{
"productName": "Cute DONUT STUD EARRINGS - Small Donut Jewelry - Donut Earrings - Earrings for girls",
"productShopName": "DesJoliesMomes",
"shopLink": "https://www.etsy.com/shop/DesJoliesMomes",
"productNoOfReviews": "954",
"productRating": "4.9406",
"productPrice": "$8.23",
"productOriginalPrice": "",
"productItems": [],
"productReviews": [
{
"feedbackRating": "5",
"feedbackText": "gli orecchini sono davvero bellissimi ricevuti subito e in una scatoletta meravigliosa le titolari sono davvero gentilissime e disponibili ho effettuato un altro ordine una settimana dopo li trovo bellissimi e oltretutto sono anallergici strepitosi",
"feedbackReviewer": "Katy Pappalettera",
"feedbackDate": "Apr18,2018"
},
{
"feedbackRating": "5",
"feedbackText": "these were so cute my niece loved them and this shop was so wonderful when i realized i d made a mistake and sent my whole order out again fabulous customer service adorable jewelry",
"feedbackReviewer": "heavysouldarlin",
"feedbackDate": "Feb18,2018"
},
{

```
            "feedbackRating": "5",
            "feedbackText": "tr s belles boucles d oreilles merci aux cr atrices pour leurs produits et leur petite attention qui accompagnait ma commande",
            "feedbackReviewer": "Emilie",
            "feedbackDate": "Feb8,2020"
         },
         {
            "feedbackRating": "5",
            "feedbackText": "",
            "feedbackReviewer": "serena",
            "feedbackDate": "Dec27,2018"
         }
      ],
      "categories": [
         "Jewelry",
         "Earrings",
         "Stud Earrings"
      ]
   }
```


Иако зборуваме за JSON сепак MongoDB во позадина работи со BSON што преставува Binary JSON. Бинарниот формат го проширува JSON типот во одредени сегменти. Пример овозможува подетални типови за броеви. Бинарниот формат овозможува поефикасно складирање доколку се гледа од аспект на меморија. Проблематиката со ObjectID кое се јавува при креирање на секој документ во колекција во формат id : ObjectId(“16123hasd1273”), ваквата синтакса не е валидна и JSON форматот не ја препознава, но MongoDB може да работи со него бидејќи го преведува во BSON формат.



Atlas и PyMongo

MongoDB Atlas овозможува лесен начин да ги хостираме и менаџираме нашите податоци на cloud. Со поставувањето на базата на податоци на cloud, ни се овозможи лесен начин да имаме централно место на постоење на податоците, така што секој кој треба да манипулира со нашите податоци доволно е само да овозможиме мрежен пристап преку конекциски стринг.

За да може лесно да се комуницира со Mongo база на податоци потребно е да се користат драјвери. Тие преставуваат алтернатива на mongo shell преку кои може да се поврземе кон MongoDB Atlas, да ги внесеме податоците и да ги извршиме сите потребни кверинја користејќи Pyhon код.

```
from pymongo import MongoClient
import json

client =
MongoClient("mongodb+srv://andrejanchevski:nebidimrdnat123@etsydatacluster.n41nn.mongodb.net/et
sydata?retryWrites=true&w=majority")
db = client.etsydata
products = db.products
with open('products_preprocessed.json', encoding='utf8') as filehandler:
    productData = json.load(filehandler)
products.insert_many(productData)
shop_owners = db.shop_owners
with open('shop_owners_crawled.json', encoding='utf8') as filehandler:
    shopOwnersData = json.load(filehandler)
shop_owners.insert_many(shopOwnersData)
categories = db.categories
with open('categories_crawled.json', encoding='utf-8') as filehandler:
    categoryData = json.load(filehandler)
categories.insert_many(categoryData)
shops = db.shops
with open('shops_preprocessed.json', encoding='utf-8') as filehandler:
    shopsData = json.load(filehandler)
shops.insert_many(shopsData)
```

Шеми на податоци и моделирање на податоци

MongoDB не присилува на никакви шеми. Документите во една колекција не мора да имаат иста шема. Документите можат да изгледаат како е потребно од самата апликација. На примерот е покажано како може во колекцијата за Продукти, да се внесат 2 документи со тотално различна структура и тип на податоци.

```
db.products({'title': "ProductA", "price": "23.50"})
db.products({'productName': "ProductA", "productPrice": "23.50"})
```

Но кога се гради одредена апликација се дава одредена минимално ограничена структура на податоците во зависност од бизнис логиката на апликацијата. Пример за да пристапуваме и филтрираме податоците мора да знаеме отприлика кои атрибути ги поседуваат и какви вредности би имале тие. Но, останува флексибилноста која ја нуди неструктурираната база на податоци за брз и ефикасен пристап.

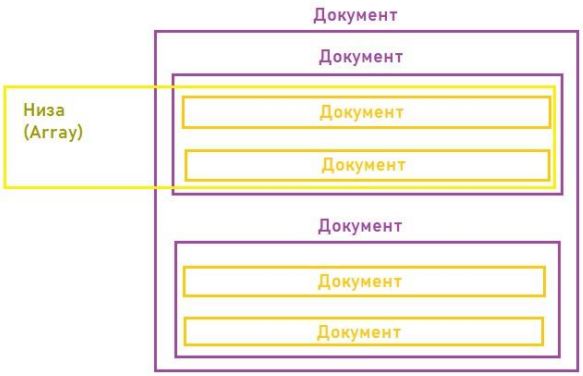
Кога се дизајнира MongoDB колекција важно е прашањето до кој степен да се оди со користењето на предефинирана шема. Постојат неколку пристапи кои покажуваат на степенот на користење на шема за документите во колекција. Едниот пристап е хаотичен каде што документите во една колекција би се разликувале целосно и во број на атрибути и по видот на атрибутите. Исто така може да се пристапи со SQL пристап каде што секој документ во една колекција ќе прати стриктна шема. Двата пристапи се можни, но не толку ефикасни, бидејќи во SQL пристапот се губи флексибилноста, а при хаотичниот пристап е многу тешко да се дизајнира соодветна апликација. Најдобриот пристап е да се земе најдоброто од двата начини, односно да постои шема во колекција која што ќе прикажува неколку задолжителни атрибути во документот, но одредени документи да може да имаат и плус дополнителни незадолжителни атрибути. Во релациона база на податоци тие колони ќе постојат во табелата и би имале null вредности, додека во MongoDB нема да постои атрибутот во документот од колекцијата и се намалуваат редундантните null вредности.

Кога се пристапува со користење на не релациона база на податоци при дизајнирањето, може да се воведат и редундантни податоци кои ќе се појавуваат во повеќе колекции и документи така што би се овозможил многу брз пристап до нив. Дополнително, доколку во некоја релациона база на податоци некои табели се тесно поврзани и треба да се прави join за да се спојат тие податоци, што претставува скапа операција, во MongoDB тие табели може да се претстават како колекција во која документите би се вгнездиле едни во други во зависност од бизнис логиката.

Релации

Постојат два начини на претставување на поврзаности помеѓу колекциите во MongoDB:

- Вгнездени Документи
- Референци



One-to-One relationship – најчесто при овој тип на релација се оди со пристап на вгнездени документи, но секако не сме присилени на тоа и може да се претстави преку референца. Се зависи од потребата на апликацијата, затоа изборот на пристап на преставување на релација е application driven.

One-to-Many relationship

Many-to-Many relationship – Во SQL, релациите се креираат на тој начин што потребно е да се креираат 3 табели каде што третата табела е меѓу табела која ги поврзува двата ентитети. Во MongoDB таквите релации можат да се направат користејќи само две колекции.

```
db.shops.aggregate([{$lookup : {from 'products', localField: 'shopName',
foreignField:'productShopName'}}])
```

Основни CRUD операции

CREATE

Креирањето на документи во колекции во MongoDB е многу лесно и постојат повеќе методи. Структурата, како и видот на податоците не се ограничени (освен ако е специфицирано при креирањето на колекција) и може да има различни документи во една колекција. Исто така, при додавањето на документи во колекцијата, доколку се изостави _id атрибутот, Монго самиот ќе креира идентификатор за документот. Овој идентификатор треба да биде уникатен и доколку се внесе документ со ист идентификатор, Монго ќе јави грешка. Овој идентификатор е од тип ObjectId, кој е со големина од 12 бајти (4 бајти за TimeStamp, 5 бајти за рандом вредност и 3 бајти за бројач). Ова овозможува кога ќе се испраша колекцијата доколку не се наведе услов за сортирање, документите ќе бидат сортирани во опаѓачки редослед според TimeStamp од ObjectId. Иако Монго може да го креира овој идентификатор автоматски, може и ние да го дефинираме со тоа што ќе зададеме некоја уникатна вредност на атрибутот _id.

На следните два примери може да се види како се искористени двата методи за креирање на документи insertOne() и insertMany() може да се креира еден документ во колекцијата Products.

```
MongoDB server version: 4.4.3
MongoDB Enterprise atlas-qmscci-shard-0:PRIMARY> db.products.insertOne({productName: "Cute T-Shirt", productShopName: "Tshirt Shop", shopLink:
"https://www.etsy.com/shop/AnniDesignsllc", productRating: 4.250, productPrice: "$25", productItems: ["Red tshirt L", "Red tshirt M", "Green ts
hirt M", "Green tshirt L"], productReviews: [{feedbackRating: 5, feedbackText: "Nice tshirt", feedbackDate: "Jun25,2020"}]});
{
        "acknowledged" : true,
        "insertedId" : ObjectId("601b0dd188670440c1af8d0d")
}
MongoDB Enterprise atlas-qmscci-shard-0:PRIMARY> db.products.insertMany([{productName: "Cute T-Shirt", productShopName: "Tshirt Shop", shopLink
: "https://www.etsy.com/shop/AnniDesignsllc", productRating: 4.250, productPrice: "$25", productItems: ["Red tshirt L", "Red tshirt M", "Green
tshirt M", "Green tshirt L"], productReviews: [{feedbackRating: 5, feedbackText: "Nice tshirt", feedbackDate: "Jun25,2020"}]}, {productName: "C
ute T-Shirt", productShopName: "Tshirt Shop", shopLink: "https://www.etsy.com/shop/AnniDesignsllc", productRating: 4.250, productPrice: "$25",
productItems: ["Red tshirt L", "Red tshirt M", "Green tshirt M", "Green tshirt L"], productReviews: [{feedbackRating: 5, feedbackText: "Nice ts
hirt", feedbackDate: "Jun25,2020"}]}, {productName: "Men Jeans", productShopName: "Jeans Shop", shopLink: "https://www.etsy.com/shop/AnniDesign
sllc", productRating: 4.250, productPrice: "$25", productItems: ["Jeans S", "Jeans M", "Jeans L"], productReviews: [{feedbackRating: 5, feedbac
kText: "Nice jeans", feedbackDate: "Jun25,2020"}]}]);
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("601b1197b2db2017249f066c"),
                ObjectId("601b1197b2db2017249f066d"),
                ObjectId("601b1197b2db2017249f066e")
        ]
}
```

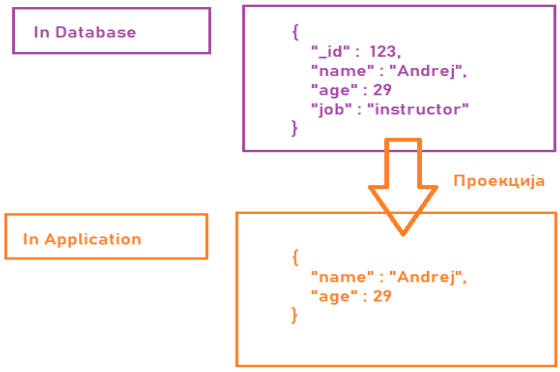
READ

Важно нешто кога работиме во shell е кога правиме всушност find операција на одредена колекција, оваа операција не ни ги враќа сите податоци, туку како резултат добиваме Cursor објект со 20 документи иако посакуваното и замислено сценарио е find да враќа низа од документи. Но доколку работеше на тој начин може да се случи сценарио каде што find ќе ни врати милион Документи и тоа би траело доста долго време.

Cursor објектот преставува објект со meta податоци кој ни овозможува да заокружиме околу резултатот. Со Cursor објектот може да користиме функции од типот на toArray() кој би ни ги дал сите Документи во една низа. Или пак it командата која би ни дала друга порција од податоците на тој Cursor објект добиен со одредена find операција.Врз Cursor објектот можеме да користиме и forEach() функција која ќе ни овозможи да изминеме низ сите документи и да направиме нешто со нив. Друг метод кој го нуди Cursor објектот е pretty() со кој документите може да се прикажат во убав json формат.

PROJECTION

Генерално може да се земат сите податоци од една колекција и да се предадат на апликацијата, но најчесто не сите податоци од документите сакаме да ги пратиме на апликацијата. Пренесувањето на податоци од база до апликација сепак на еден начин преставува пренесување податоци низ мрежа. Мрежата нема неограничени ресурси, има одреден bandwidth и пренос на голем број на податоци може да доведе до голема латентност. За да може да се заштитиме од такви сценарија се користат проекции.




```
> db.products.find({categories: { $all: ["Jewelry", "Earrings"] }, "productReviews.feedbackRating" : { $lte: 4 }},
  { _id: 0, productName: 1, productShopName: 1, productPrice: 1}).sort({"productPrice": 1}).limit(30)
```

UPDATE

Главни методи за едитирање на документите претставуваат updateOne(filter, update, options), updateMany(filter, update, options) и replaceOne(filter, update, options).

- filter - претставува условот за одбирање на документите кои ќе се едитираат.
- update - претставува логиката за едитирање на документот.
- options - тука можат да се наведат сите опции за начинот на едитирање на документот. Една од нив претставува upsert кој ако се постави, доколку филтерот не пронајде некој документ во колекцијата, според логиката за едитирање ќе се креира нов документ.

На следниот пример е искористена forEach функцијата која ја нуди Cursor објектот, каде што за секој сопственик на продавница типот на атрибутите ownerFollowers и ownerFollowing од String се парсира во Int тип.

```
MongoDB Enterprise atlas-qmscci-shard-0:PRIMARY> db.shop_owners.find().forEach(function(data){ db.shop_owners.updateOne({ "_id"
: data._id },{ "$set": { "ownerFollowers": parseInt(data.ownerFollowers), "ownerFollowing" : parseInt(data.ownerFollowing) } })
})
```

На овој пример може да се види како се земаат сите продукти кои имаат број на оставени коментари поголем од 1000 и рејтинг поголем од 4.5 и се додава атрибут discount со вредност 5 долари. $set операторот доколку дефинираниот атрибут не постои во документот, соодветно ќе го креира.

```
MongoDB Enterprise atlas-qmscci-shard-0:PRIMARY> db.products.updateMany({
"productNoOfReviews": { $gt: 1000}, "productRating": { $gt: 4.5}}, { $set:
 {"discount": "$5"} } );
```

Валидација на шема

Главна карактеристика на MongoDB е флексибилноста, а тоа е овозможено преку можноста да имаме различни шеми на документи во една иста колекција, но некогаш сакаме да ја заклучиме флексибилноста поради бизнис логиката на апликацијата за која ја дизајнираме базата на податоци.

Тоа ограничување можеме да го постигнеме со валидација на шема. Најчесто имаме додавање, или промена на податоци во една колекција. Валидацијата на шема ќе ни помогне да ги валидираме податоците кои треба да се додадат во една колекција и валидацијата ќе се основа на наша дефинирана шема. Со тоа валидацијата всушност прави дозвола или забрана на додавање податоци во дадена колекција.

Со валидација на шемата можеме да дефинираме какви видови на операции сакаме да валидираме (validationLevel) и што сакаме да направиме кога ќе падне валидацијата (validationAction).

- validationLevel – кој документи да ги валидираме
  - strict – сите insert и update операции се валидираат
  - moderate – се валидираат сите insert операции но update операциите се прават на документи кој биле валидни претходно
- validationAction – што ќе се случи доколку падне валидација
  - error – да фрлиме грешка и да го одбиеме запишувањето или ажурирањето на податоци
  - warn – да логираме предупредување но да се изврши запишувањето или ажурирањето

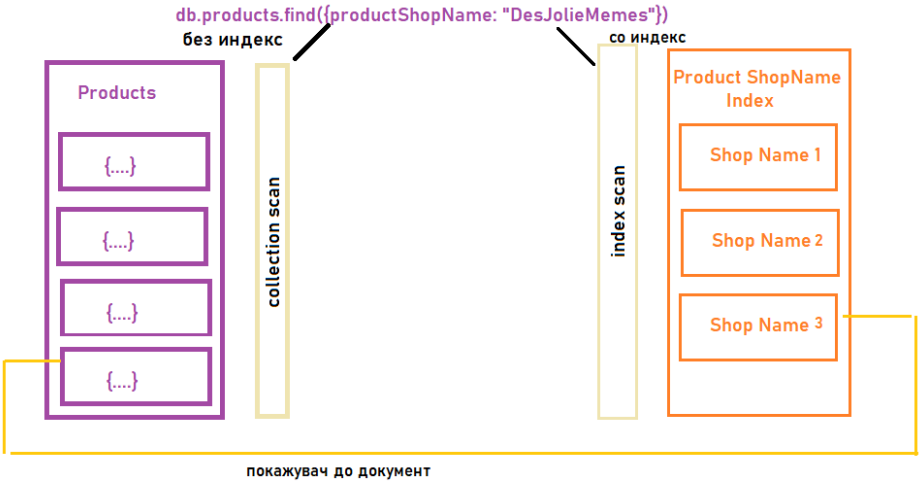
```
db.createCollection('products',{
  validator: {
    $jsonSchema: {
      bsonType: 'object',
      required: ['productName','productShopName','productPrice','productReviews'],
      properties: {
```

```
      productName: {
        bsonType: "string",
        description: "must be a string and is required"
      },
      productPrice: {
        bsonType: "number",
        description: "must be an mnumber and it is required"
      },
      productReviews: {
        bsonType:'array',
        description: "must be an array and is required",
        items: {
          bsonType: object,
          required: ['feedbackRating','feedbackDate']
        }
      }
    }
  }
 }
})
```

Индекси

Индексите ни помагаат на таков начин што ги забрзуваат find, update или delete кверињата, односно сите квериња со кои што пребаруваме одредени документи кои соодветствуваат на одреден критериум.

Доколку не поставиме индекс на критериумот по кој правиме пребарување во тој случај MongoDB ќе направи скенирање на целата колекција, односно за да го изврши кверито ќе ја измине целата колекција, односно секој нејзин документ за да види дали условот е исполнет. Во база на податоци со голем број документи овој процес ќе потрае.



Индекс не претставува замена за колекција, туку дополнување на колекцијата. Кога се креира индекс, го креираме за одреден атрибут на документот во примерот shopName. Така креираниот индекс постои дополнително за колекцијата и претставува подредена листа од сите вредности на атрибутот за кој е креиран индексот од сите документи во колекцијата. Секој елемент од индексот кој претставува вредност на атрибутот за кој е креиран содржи покажувач до неговиот документ. Со креирањето на индекс можеме да направиме така наречено индексно скенирање. Кога MongoDB прави индексно скенирање се движи низ вредностите на тој атрибут и кога ќе ја најде посакуваната вредност скока до документот користејќи го покажувачот. Поради подреденоста не мора да ги поминува сите индекси, туку знае како се подредени и знае точно низ кој дел од индексот да пребарува.

Иако индексите носат одредена ефикасност на базата, лоша практика е да се прават индекси за секој можен атрибут. Иако креирањето на индекс за секој атрибут ќе го забрза процесот на пребарување, сепак индексите си имаат свој трошок во однос на перформанси. Со користење на голем број индекси цената на перформанси ќе се плати во insert операциите, бидејќи со секое внесување во документација потребно е да се внесе запис во индексот која е подредена листа што значи дека новиот запис треба правилно да се смести во индексот. Како се зголемува бројот на индекси, така insert операцијата ќе бара повеќе перформанси.

```
db.shop_owners.find({"ownerFollowing" : {$gt : 500}})
db.shop_owners.explain().find({"ownerFollowing" : {$gt : 500}})
```

Кога при кверијата за пребарување ја користиме explain() методата во тој случај MongoDB ни дава информации за winingPlan или rejectedPlan. Доколку не користиме индекси во тој случај секако имаме еден начин на извршување на кверито, а тоа е целосно пребарување низ колекцијата. Доколку се користат индекси во тој случај може да видиме дали со користењето на индекси имаме поголема ефикасност. Доколку на explain методот додадеме аргумент explain(“executionStats”) во тој случај можеме да забележиме и повеќе метрики и тоа: за колку време ни се извршило кверито и дали е направен collection scan или index scan, доколку сме користеле индекси колку клучеви се посетени и сите статистички информации поврзани со операцијата на пребарување.

```
MongoDB Enterprise atlas-qmscci-shard-0:PRIMARY> db.shop_owners.createIndex({"ownerFollowing":1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "commitQuorum" : "votingMembers",
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1612388731, 29),
                "signature" : {
                        "hash" : BinData(0,"c79rdaKp3DLvs29VH8y7FbgqmGw="),
                        "keyId" : NumberLong("6924374514021171203")
                }
        },
        "operationTime" : Timestamp(1612388731, 29)
}
```

Кога креираме single field индекс специфицираме за кој атрибут го креираме и дали сакаме индексот да се креира во растечки или опаѓачки редослед.

Кога нашето пребарување ќе таргетира поголемо количество на документи во колекцијата во тој случај користењето на индекси ќе е побавно од пребарување без индекси, бидејќи кај индексите MongoDB мора да го направи чекорот каде што од записите во индексот се префрла кон колекцијата да го земе соодветниот документ.

Compound Index - индекс кој што во себе содржи повеќе полиња. По поставувањето на кои полиња сакаме да индексираме нема да се креираат два индекси, туку ќе се креира еден индекс со комбинирани вредности. Така секој елемент во индексот ќе е всушност множество од вредностите на специфицираните атрибути кога се гради индексот. Исто така важен е редоследот на специфицирање на атрибутите кога се гради индекс, бидејќи и во тој редослед вредностите во записите ќе се поставени.

```
MongoDB Enterprise atlas-qmscci-shard-0:PRIMARY> db
etsydata
MongoDB Enterprise atlas-qmscci-shard-0:PRIMARY> db.shops.createIndex({"currentItems": 1, "shopName": 1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "commitQuorum" : "votingMembers",
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1612392426, 31),
                "signature" : {
                        "hash" : BinData(0,"oqqMncZg06C4mE6rUwixe2WJWr8="),
                        "keyId" : NumberLong("6924374514021171203")
                }
        },
        "operationTime" : Timestamp(1612392426, 31)
}
MongoDB Enterprise atlas-qmscci-shard-0:PRIMARY>
```

Индексите исто така можат да ни помогнат и при сортирање. Самата карактеристика на индексот дека записите се сортирани ќе го прави сортирањето на документи поефикасно. Доколку не користиме индекси и направиме сортирање на голем број документи, MongoDB може да ни направи timeout бидејќи има во себе 32MB threshold да земе во себе и да ги сортира.

```
db.shop_owners.find({}).sort({"ownerFollowing": 1})
```

Default Index - MongoDB стандардно во себе содржи default индекс кој што работи по едно поле, а тоа е ID. Доколку филтрираме и сортираме по ID во тој случај имаме автоматско оптимизирање на тие процеси.

```
db.showIndexes() - ги дава сите тековни индекси на базата
```

Aggregation Framework

За да можеме да постигнеме оптимизација на прашалник во однос на подобрени перформанси се користат агрегациони операции кои што ја делат комплексноста на полесни сцени со засебни операции. Во еден прашалник бројот на сцени не е ограничен.

Доколку ги гледаме сцените како еден pipeline се започнува со $match така што се селектираат Документите со кои сакаме да работиме. $sort ги сортира исфилтрираните документи според вредноста на одредената поле. Со поставувањето на $sort после $match го минимизираме бројот на објекти кои треба да се сортираат бидејќи сортирањето е скапа операција. Сумаризација на прашалниците можеме да направиме со следниот чекор од pipeline со помош на $group. Доколку не сакаме да го вратиме целиот документ се користи $project каде што прецизираме кој атрибути сакаме да се вратат од документот.


```
> db.products.aggregate([{
  $group:
  {
  _shopName: "$productShopName",
  avgRating: { $avg: "$productRating" },
  noOfRatings: {$sum: "$productNoOfReviews"}
  }
  }, {
  $sort:
  {
  avgRating: -1
  }
  }, {
  $lookup: {
  from: "shops",
  localField: "productShopName",
  foreignField: "shopName",
  as: "shop_info"
  }},
  {
  $project: {
  _shopName: 1,
  avgRating: 1,
  noOfRatings: 1
  }
  }
 ]).pretty()
```