ext_aeid5561-Ask - DB Engines

Analysis & Design Document

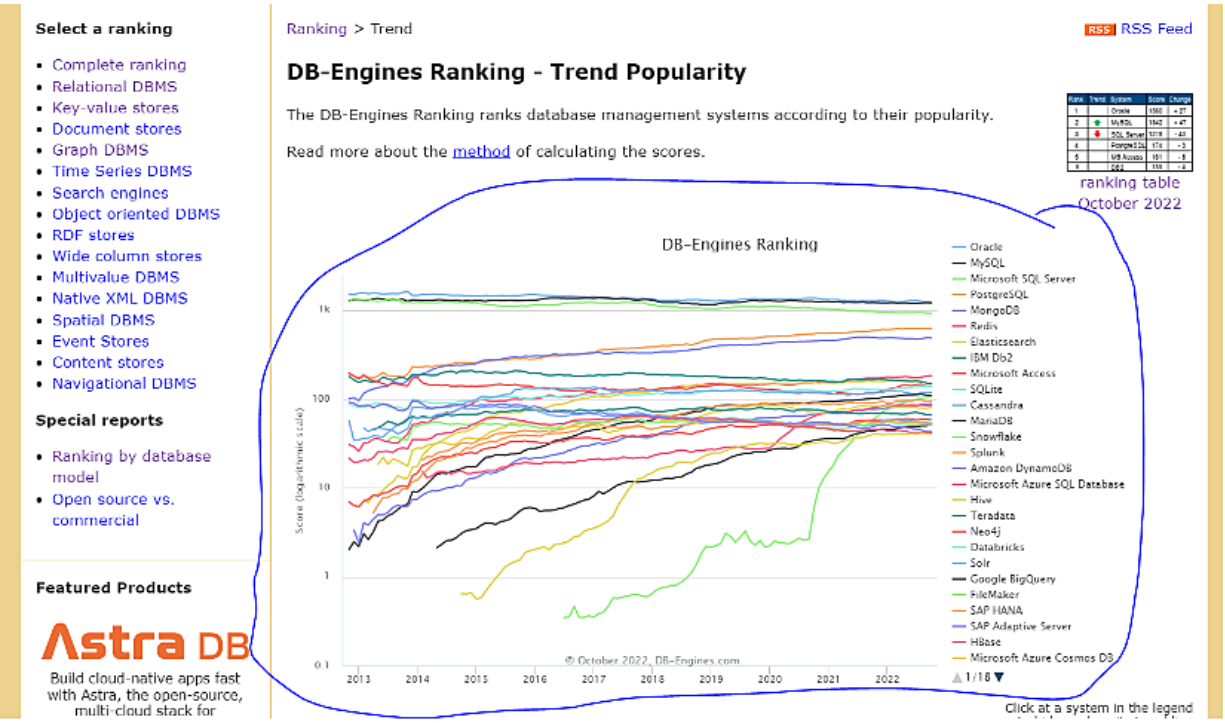
Date – 26 Oct 2022

Table of Contents

Topic **Page No.**

1. Scope of work

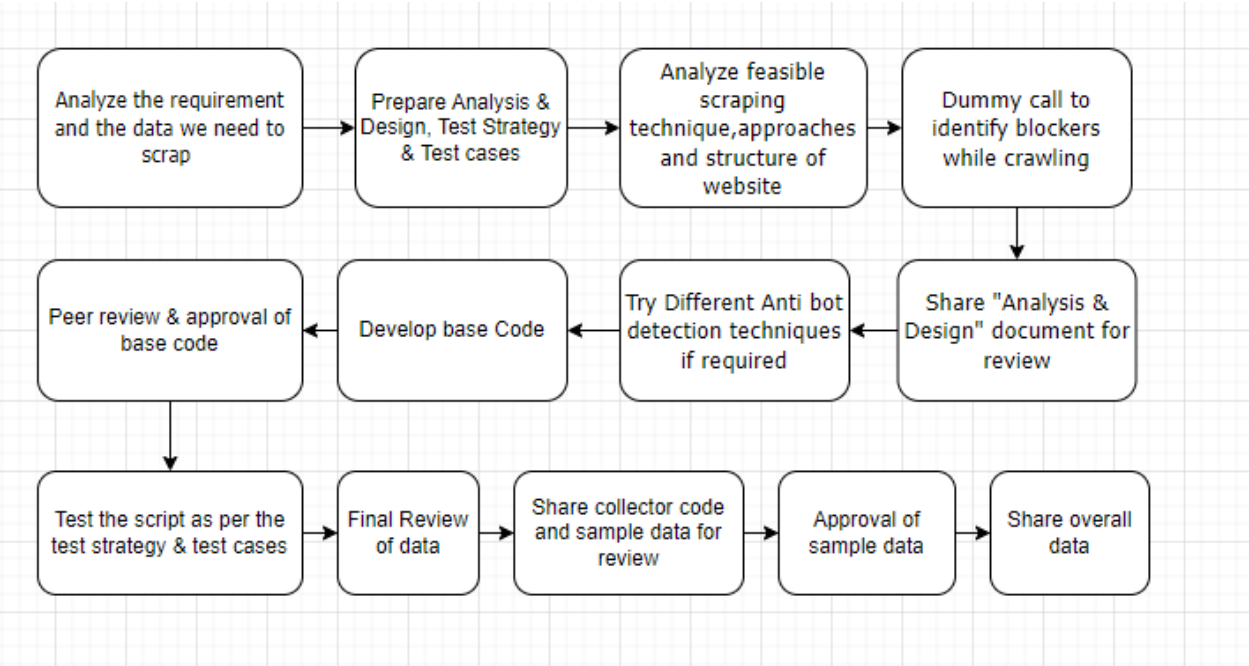
Collect trend popularity scores across all products, and all history. Monthly data should go back to 2013, with potential expected records of ~58K (12 months/yr, 10 years, ~486 products) for SITE: https://db-engines.com/en/ranking_trend



2. Solution Approach

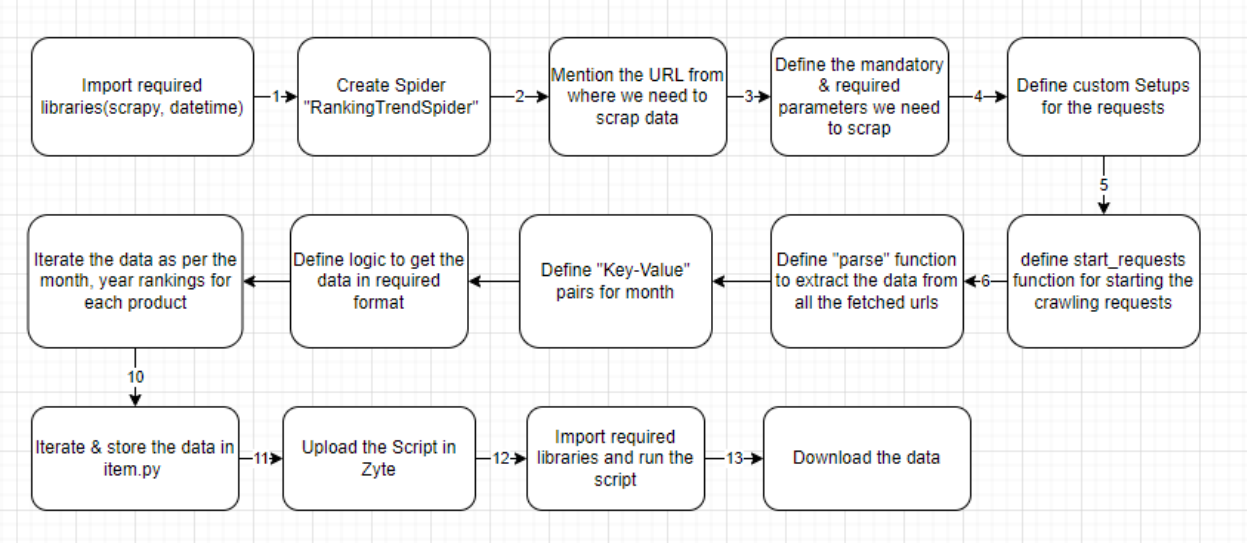
We are following the below steps to develop the script as per the requirement

- The website is **global**, hence only one collector code is needed.
- We are fetching the rankings for each product from Nov 2012 to the current month.
- Checked the javascript data (the data we get from AJAX calls) with the help of view page source.
- Scrapped the script data by using XPath.
- Trimmed the extra data that is not useful and then created dictionary.


Analyze the requirement and the data we need to scrap
Prepare Analysis & Design, Test Strategy & Test cases
Analyze feasible scraping technique,approaches and structure of website
Dummy call to identify blockers while crawling
Share "Analysis & Design" document for review
Try Different Anti bot detection techniques if required
Develop base Code
Peer review & approval of base code
Test the script as per the test strategy & test cases
Final Review of data
Share collector code and sample data for review
Approval of sample data
Share overall data

3. Script Development Flow

Below steps are followed to create spider



4. Technology Considerations

Custom signup - Not required

Programming Language - Python

Framework - Scrapy

Tool - Zyte

Functions & Libraries used - datetime, scrapy-user-agents

Storage (Database) - Zyte Cloud

Deployment Requirements

- Install all the required libraries in Zyte Cloud

Logging considerations

- No logging is required
- No CAPCTHA authentication required

Proxy Details

We are using user agent to avoid getting blocked, this is present in settings.py file.

5. Base Collector Code

File name - ranking_trend.py

Here we are scraping the required and mandatory data

Step 1 - Importing libraries here

```
import scrapy
import json
import datetime
from ..items import DbEnginesItem
```

Step 2 - Here a spider named "RankingTrendSpider" is created

```
class RankingTrendSpider(scrapy.Spider):
    name = 'ranking_trend'
```

Step 3 - Here allowed domain and start url of the website are defined that we are crawling

```
    allowed_domains = ['http://db-engines.com/']
    start_urls = f"https://db-engines.com/en/ranking_trend"
```

Step 4 - Here all Mandatory Fields Data are defined under the main class that will be called using "self."

```
    context_identifier = "DB-Engines"
    execution_id = ""  #This will be taken automatically from zyte, for now this is hardcoded
    feed_code = "aeid5561"
    record_create_by = "aeid5561_ranking_trend"
    record_create_dt = datetime.datetime.utcnow().strftime('%Y-%m-%d %T')
    site = "https://db-engines.com/en"
    source_country = "Global"
    src = "https://db-engines.com/en/ranking_trend"
    type = "Product Ranking"
```

Step 5 - Here we are defining custom settings that are needed for crawling

```
        custom_settings = {
            'ROBOTSTXT_OBEY': False,
            'COOKIES_ENABLED': True,
            'COOKIES_DEBUG': True,
```

```
    'AUTOTHROTTLE_ENABLED': True,
    'DOWNLOAD_TIMEOUT': 20,
    'DUPEFILTER_DEBUG': True,
}
```

Step 6 - Here we are defining start_requests function for starting the crawling requests

```
def start_requests(self):
    yield scrapy.Request(url=self.start_urls, callback=self.parse)
```

Step 7 - Here we are defining parse function. Inside this function we are writing code for crawling the data

```
def parse(self, response):
    item = DbEnginesItem()  # Object to store data in items.py
    data = response.xpath('//script[@type="text/javascript"]/text()').get()     #Here we are fetching script data in which all the ranking data are present
```

Step 8 - Here we are Removing extra data coming at front and last of the script

```
    str1, str2 = 'var dbe_data = [', 'var dbe_title'
    idx1, idx2 = data.index(str1), data.index(str2)
    data1 = data[idx1 + len(str1) + 1: idx2]
    data1 = data1[:-1]
```

Step 9 - Here we are manipulating and replacing some data to convert the coming script data from str to dictionary

```
    data1 = data1.replace("null", "0").replace("data", """data""").replace("name", """name""").replace("visible", """visible""").replace("false", """false""")
    data1 = data1.replace("""Tera"data" Aster""", """Teradata Aster""").replace("""1010"data""", """1010data""").replace("""Tera"data""", """Teradata""")
```

Step 10 -Here we are splitting data to convert it from string into list

```
    data2 = data1.split("},")
    print("data2=====", data2)
```

Step 11 - Here we are passing empty lists for using it in further code to append data

```
    ranking = []
    name = []
```

```
visible = []
lst = []
months = []
```

Step 12 - Here we are defining one list for the month data which will contain all the months name

```
list_month = ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October",
              "November", "December"]
```

Step 13 - Here we are using for loop and doing some manipulations for converting data coming from script to dictionary

```
for d in range(len(data2)):
    data3 = data2[d].rstrip("}]")
    data3 = data3 + "}"
    data3 = json.loads(data3)
    print("data3=====", data3)
    lst.append(data3)          # Here we are appending dictionary data to create a list of dictionaries
```

Step 14 - Here from this for loop we are fetching all the data that are required from website by going one by one inside dictionaries

```
for dict_1 in range(len(lst)):
    for i, j in lst[dict_1].items():          # Here we are iterating for key value pairs
        year = 2012                           # Here we are initializing start of year and month from "november 2012" according to website
        month = 11

        if i == "data":
            for k in lst[dict_1][i]:          # Here loop is iterating for all the ranking data present in list and then giving raking, month, visible and name data

                if k == 0:                    # for fetching raking data
                    ranking.append("Null")
                else:
                    ranking.append(k)

                if month == 13:               # for fetching month and year data
                    year += 1
                    month = 1
```

```
            months.append(f"{list_month[month - 1]} {year}")

            name.append(lst[dict_1]['name'])        # for fetching name data
            month += 1

            if lst[dict_1].get("visible") == 'false':       # for fetching visible data
                visible.append("No")
            else:
                visible.append("Yes")
```

Step 15 - Here we are using for loop for storing all items data one by one in items.py and some data are taken from self. because we have defined it in the main class

```
        for i in range(len(name)):
            item["Name"] = name[i]
            item["Month"] = months[i]
            item["Ranking"] = ranking[i]
            item["Visible"] = visible[i]
            item["Context_identifier"] = self.context_identifier
            item["Execution_id"] = self.execution_id
            item["Feed_code"] = self.feed_code
            item["Record_create_by"] = self.record_create_by
            item["Record_create_dt"] = self.record_create_dt
            item["Site"] = self.site
            item["Source_country"] = self.source_country
            item["Src"] = self.src
            item["Type"] = self.type
            yield item                              # yielding all items here
```

6. Template Parameters & Description

The template contains the data that is scraped as per the ranking of newly listed products.

For the parameters where **mandatory** is mentioned, this is mandatory parameters as per the required template.

For the parameters where **Required** is mentioned, this is parameters needed as per the requirement document.

Below are the parameters that we are scraping and their description

1. **key -** Zyte by default add this as an identifier.
2. **Context_identifier (Mandatory) -** Currently we don't have any breadcrumbs for the website so hardcoded this.
3. **Execution_id (Mandatory) -** Execution id will be taken automatically from zyte, for now this is hardcoded as we are not using pipelines.
4. **Feed_code (Mandatory) -** This is hardcoded as project name.
5. **Month (Required) -** This will have month and year data.
6. **Name (Required) -** This is the name of the product.
7. **Ranking (Required) -** This is the ranking value as per the website.
8. **Record_create_by (Mandatory) -** This is hardcoded with spider name
9. **Record_create_dt (Mandatory) -** This is the timestamp for capturing the data.
10. **Site (Mandatory)-** This is hardcoded.
11. **Source_country (Mandatory) -**This is hardcoded as per the project code.
12. **Src (Mandatory) -** This is the link for product details page.
13. **Type (Mandatory) -** This is hardcoded.
14. **Visible -** This will tell us if the product trend is visible or not.

7. Risks and Dependencies

Below are the identified risks and their possible solutions:

Risk	Mitigation
Risk of getting blacklisted/blocked/IP restrictions due to security/network policies on the web server.	We need to control the concurrency & use different proxy methods.
If the semantic code/markup of the website changes, the script will have a possibility of failure.	Identify the changes in the semantic code/markup of the website and modify the script accordingly.