# Web Crawling with Maximum Depth

**1.Objective:**

The goal is to create a web crawler that starts from the given URL (<https://navbharattimes.indiatimes.com/>) and collects data up to a specified maximum depth. The crawler will fetch the URLs, titles, and content of articles while adhering to filters, depth, threading, and storing data in a database.

**2. Approach**

1. Understanding Requirements:
   * Start crawling from the root URL.
   * Use a regex filter to identify valid child URLs for consistency and relevance.
   * Use threading to enhance efficiency by managing multiple requests concurrently.
   * Stop crawling beyond the specified depth level.
   * Extract article titles and content using XPath expressions.
   * Handle errors and ensure data storage mechanism.
2. Usage of Crawl Queue:
   * The crawl queue is a critical component of any web crawler, as it determines the order in which pages are fetched and parsed.
   * Initialize a crawler by creating a queue(Linked lists)
   * Fetch and parse urls using HTTP Client and JSoup
   * Use regex filter and filter urls and continue the process until the specified maximum depth is reached.
3. Using Regex filter
   * Create a regex pattern that matches the structure of valid child URLs for the target website.
   * Use a HTML Parser (JSoup) to extract the <a> tags
   * Then apply the regural expression filter to find out invalid and valid child url links
   * Add them to the crawl queue.
4. Execution Steps:
   * Step 1: Fetch the page content of the starting URL using HTTP requests.
   * Step 2: Parse the HTML response using an HTML parser like Jsoup (Java),
   * Step 3: Apply a regex filter to extract valid child URLs.
   * Step 4: Recursively crawl the child URLs, respecting the maximum depth and threading constraints.
   * Step 5: Extract article titles and content using the provided XPath expressions.
   * Step 6: Store the crawled data (URL, title, content, depth, and timestamp) in a database for retrieval and analysis.

# 

# **3. Logic**

# Crawler Logic:

* + Start with a queue containing the root URL and its depth (initial depth = 0).
  + Fetch and parse the page content of the URL at the front of the queue.
  + Apply regex to identify valid child URLs.
  + If the current depth is less than the maximum depth, add child URLs to the queue with their incremented depth.
  + Extract relevant content using XPath expressions and store it in the database.
  + Use threading to process multiple URLs concurrently.

1. XPath Logic:
   * XPath is a query language for selecting nodes (elements, attributes, etc.) from an XML or HTML document.
   * Fetch the Webpage using HTTP Client
   * Parse the HTML content with JSoup
   * Write XPath expressions to locate the desired element or table.
   * Extract data from the parsed HTML document using XPath expressions.

### 

### 

### 

### **4. Tech Stack**

1. Programming Language:
   * Java (e.g., Jsoup, Queue Crawling).
2. Database:
   * MySQL for storing crawled data.
   * Xpath language(xml)
3. Tools:
   * Regex: For filtering child URLs.
   * HTTP Client : for requesting and receiving response from webserver.

**5. Database Design**

| Column name | Data type | Description |
| --- | --- | --- |
| Id | Int primary key | Unique identifier |
| Url | varchar(50) | Url of the page |
| Article content | Text | Extracted content |
| Article title | Text | Extracted title |
| Depth level | Int | Depth level |

**6. Database query:**

CREATE TABLE crawled\_data (

id INT PRIMARY KEY,

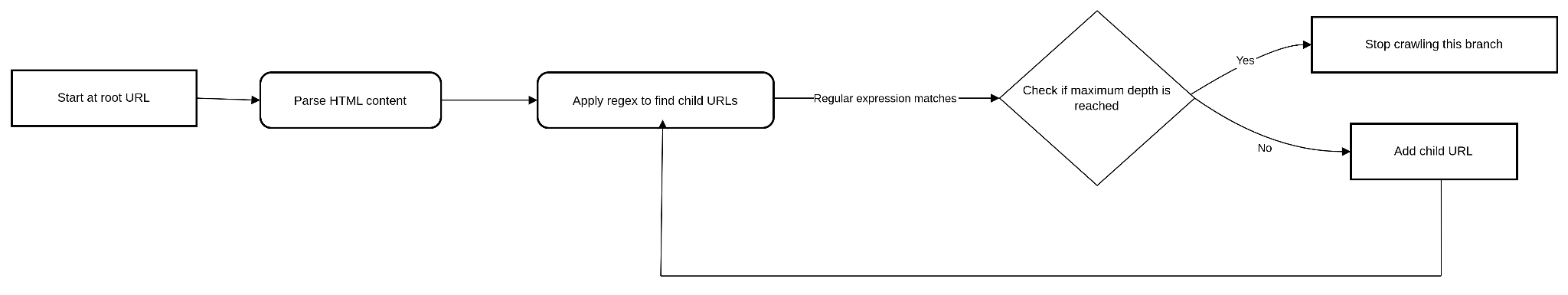
url VARCHAR(50) NOT NULL,

article\_title TEXT,

article\_content TEXT,

depth\_level INT NOT NULL);

**7. Flow Chart**



**8 Conclusion**

Building an efficient web crawler requires careful planning to balance performance, scalability, and ethical web scraping practices. By using techniques like regex filtering,Crawl Queue and HTML parsing, we can systematically extract valuable information while keeping the process efficient and organized.

A well-structured crawl queue helps manage URLs efficiently, avoiding redundant visits and optimizing resource usage. JSoup makes HTML parsing straightforward, while regex helps filter and extract relevant links.

Storing the extracted data in a database allows for easy retrieval, analysis, and reporting. With a well-designed crawler, we can automate data collection from websites.