Elasticsearch Guild

Analyzers & Edge N-gram Token Filter

Halil Bülent Orhon

Agenda

1. What is an inverted index?
2. What is an analyzer?
3. Built-in and custom analyzers
4. What is the Wildcard query?
5. What is the Edge N-gram tokenizer
6. Wildcard vs Edge N-gram

What is an Inverted Index?

- Maps **terms** to **document IDs**
- Enables fast full-text search

```
Term     → DocIDs
-----------------
cikolatali → 1, 3, 8
ve         → 1, 2, 5
fistikli   → 1
kek        → 1, 7
```

What is an Analyzer?

- Analyzer = `char_filter` → `tokenizer` → `token_filter`
- It transforms raw text into searchable tokens.
- Crucial for building the **inverted index**

Analyzer Usage: Index-time vs Query-time

- Analyzers are applied at **two points**:
  - **Index-time** → When documents are stored
  - **Query-time** → When user searches

⚠️ **Mismatch** between them can lead to no results!

Analyzer Component Summary

Component	Responsibility
`char_filter`	Pre-tokenization text normalization (e.g. replace `&` with `and`)
`tokenizer`	Splits text into tokens (e.g. words)
`token_filter`	Transforms tokens (e.g. lowercase, stemming, n-gram)

🧠 These three components form a complete **analyzer**

→ Customized analyzers = better search quality and control

Analyzer Pipeline – Step-by-Step Example

Input: `"Çikolatalı & fıstıklı kek"`

1. **Char Filter**
   `"&"` → `"ve"`
   → `"Çikolatalı ve fıstıklı kek"`
2. **Tokenizer** (`standard`)
   → `["Çikolatalı", "ve", "fıstıklı", "kek"]`
3. **Token Filters** (`lowercase`, `asciifolding`)
   → `["cikolatali", "ve", "fistikli", "kek"]`

✅ Final tokens are stored in the inverted index

Some Built-in Analyzers

Analyzer	Description / Best Use Case
`standard`	General-purpose full-text search (default)
`simple`	Clean, letter-only content
`whitespace`	Tokenize without affecting punctuation
`keyword`	Exact-match fields
`pattern`	Custom delimiter-based tokenization

What is the `stop` Token Filter?

- Removes common words like `"and"`, `"or"`, `"the"`
- Helps reduce index size and noise

Example:

Input: `"çikolatalı ve fıstıklı kek"`
Output after `stop`: `["çikolatalı", "fıstıklı", "kek"]`

⚠️ Removing stopwords may break **phrase queries**

What is `word_delimiter` Token Filter?

Input: `"blue-jeans123"`

Output: `"blue"`, `"jeans"`, `"123"`, `"bluejeans123"`, `"blue-jeans123"`

```json
"wordDelimiterTokenFilter": {
  "type": "word_delimiter",
  "catenate_all": true,
  "split_on_numerics": true,
  "preserve_original": true
}
```

Creating a Custom Analyzer

Combine tokenizer, char filters, and token filters tailored to your language and use case.

You can define it under index settings:

→ see next slide for example

Example: Simplified Turkish Analyzer

```
"analyzer": {
  "turkish_suggestion_analyzer": {
    "type": "custom",
    "tokenizer": "standard",
    "char_filter": ["turkishCharFilter"],
    "filter": [
      "lowercase",
      "asciifolding",
      "word_delimiter"
    ]
  }
},
"char_filter": {
  "turkishCharFilter": {
    "type": "mapping",
    "mappings": ["ı => i", "ş => s", "ç => c"]
  }
}
```

Wildcard Query

```
{
  "simple_query_string": {
    "fields": ["text"],
    "query": "elb*",
    "analyze_wildcard": true
  }
}
```

⚠️ Requires scanning **all terms** → Slow

Wildcard Query - Performance Issues

- Query-time operation
- High CPU usage
- Not scalable
- Grows with vocabulary size

Edge N-Gram Analyzer – What It Does

- Indexes all prefixes of a word

Input: `"elbise"`
Indexed tokens: `["e", "el", "elb", "elbi", "elbis", "elbise"]`

→ Matched directly at query-time

Why Use Edge N-Gram?

- Shifts cost to **index-time**
- Low latency at query-time
- Ideal for:
  - Search suggestions
  - Prefix search

Edge N-Gram Analyzer – Definition

```
"filter": {
  "edgeNgramFilter": {
    "type": "edge_ngram",
    "min_gram": 1,
    "max_gram": 20
  }
},
"analyzer": {
  "suggestionAnalyzer": {
    "type": "custom",
    "tokenizer": "standard",
    "filter": [
      "turkish_lowercase",
      "wordDelimiterTokenFilter",
      "asciifolding",
      "edgeNgramFilter"
    ]
  }
}
```

Search Analyzer

```
"analyzer": {
  "suggestionSearchAnalyzer": {
    "type": "custom",
    "tokenizer": "standard",
    "filter": [
      "turkish_lowercase",
      "wordDelimiterTokenFilter",
      "asciifolding"
    ]
  }
}
```

Mapping with Index + Search Analyzer

```
"mappings": {
  "properties": {
    "text": {
      "type": "text",
      "analyzer": "suggestionAnalyzer",
      "search_analyzer": "suggestionSearchAnalyzer"
    }
  }
}
```

Search Without Wildcard

```
{
  "simple_query_string": {
    "fields": ["text"],
    "query": "elb"
  }
}
```

✅ `"elb"` is already indexed

✅ No wildcard needed

Performance Results

Metric	Wildcard	Edge N-Gram	Gain
Response Time	40.3ms	15.8ms	60% faster
Search Rate	1.4k / sec	63.6k / sec	44× higher
CPU Usage (peak)	82%	31%	62% lower
Index Size	6.8 GB	7.2 GB	+5.8%

Key Takeaways

- Wildcard is slow, not scalable
- Edge N-Gram shifts work to index-time
- Huge performance benefits with tiny index cost

Conclusion

✅ Edge N-Gram Analyzer provides:

- ⚡ Faster search latency
- 🧠 Lower CPU usage
- 📈 Higher throughput
- 💰 Reduced infrastructure needs

Recommended for:

- Search suggestions
- Autocomplete
- High-traffic search boxes

Thank you for listening!