

SIMPLE QUERIES:

1. Write a query to get bank accounts that are in California:

```
get bank/_search
{
  "query": {
    "match": {
      "state": "CA"
    }
  }
}
```

```
History Settings Variables Help 200 - OK 100 ms
1 get bank/_search
2
3 get bank/_search
4 { "took": 2,
5   "timed_out": false,
6   "_shards": {
7     "total": 1,
8     "successful": 1,
9     "skipped": 0,
10    "failed": 0
11  },
12  "hits": {
13    "total": {
14      "value": 17,
15      "relation": "eq"
16    },
17    "max_score": 4.046554,
18    "hits": [
19      {
20        "_index": "bank",
21        "_id": "68",
22        "_score": 4.046554,
23        "_source": {
24          "account_number": 68,
25          "balance": 44214,
26          "firstname": "Hall",
27          "lastname": "Key",
28          "age": 25,
29          "gender": "F",
30          "address": "927 Bay Parkway",
31          "employer": "Eventex",
32          "email": "hallkey@eventex.com",
33          "city": "Shoemaker",
34          "state": "CA"
35        }
36      },
37      {
38        "_index": "bank",
39        "_id": "328",
40        "_score": 4.046554,
41        "_source": {
42          "account_number": 328,
43          "balance": 12523,
44          "firstname": "Good"
45        }
46    ]
47  }
48 }
```

2. Write a query to get bank accounts that are in California and work in Techade

```
GET bank/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "state": "CA" } },
        { "match": { "employer": "Techade" } }
      ]
    }
  }
}
```

```

1 get bank/_search
2
3 get bank/_search
4 {
5   "query":{
6     | "match":{
7       | | "state":"CA"
8     }
9   }
0 }
1
2
3 # find "Techade" accounts in CA only
4 GET bank/_search
5 {
6   "query": {
7     "bool": {
8       "must": [
9         { "match": {"state": "CA"} },
0         { "match": {"employer": "Techade"} }
1       ]
2     }
3   }
4 }

```

▶ ⏴

||

```

1 {
2   "took": 5,
3   "timed_out": false,
4   "_shards": {
5     "total": 1,
6     "successful": 1,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": {
12      "value": 1,
13      "relation": "eq"
14    },
15    "max_score": 10.552503,
16    "hits": [
17      {
18        "_index": "bank",
19        "_id": "413",
20        "_score": 10.552503,
21        "_source": {
22          "account_number": 413,
23          "balance": 15631,
24          "firstname": "Pugh",
25          "lastname": "Hamilton",
26          "age": 39,
27          "gender": "F",
28          "address": "124 Euclid Avenue",
29          "employer": "Techade",
30          "email": "pughhamilton@techade
31          .com",
32          "city": "Beaulieu",
33          "state": "CA"
34        }
35      }
36    ]
37  }

```

3. Get bank details in California and not employed in Techade

GET bank/_search

```
{
  "query":{
    "bool":{
      "must":[
        {"match":{
          "state":"CA"
        }}
      ],
      "must_not":[
        {"match":{
          "employer":"Techade"
        }}
      ]
    }
  }
}
```

```

        }
    ]
}

}

History Settings Variables Help 200 - OK 110 ms
1 get bank/_search
2
3 get bank/_search
4 {
5   "query": {
6     "match": {
7       "state": "CA"
8     }
9   }
10 }
11
12
13
14 GET bank/_search
15 {
16   "query": {
17     "bool": {
18       "must": [
19         {"match": {
20           "state": "CA"
21         }}
22       ],
23       "must_not": [
24         {"match": {
25           "employer": "Techade"
26         }}
27       ]
28     }
29   }
30 }
31

```

skipped ...
"failed": 0

8 },
 9 },
 10 "hits": {
 11 "total": {
 12 "value": 16,
 13 "relation": "eq"
 14 },
 15 "max_score": 4.046554,
 16 "hits": [
 17 {
 18 "_index": "bank",
 19 "_id": "68",
 20 "_score": 4.046554,
 21 "_source": {
 22 "account_number": 68,
 23 "balance": 44214,
 24 "firstname": "Hall",
 25 "lastname": "Key",
 26 "age": 25,
 27 "gender": "F",
 28 "address": "927 Bay Parkway",
 29 "employer": "Eventex",
 30 "email": "hallkey@eventex.com",
 31 "city": "Shawmut",
 32 "state": "CA"
 33 }
 34 },
 35 {
 36 "_index": "bank",
 37 "_id": "328",
 38 "_score": 4.046554

4. Writing a query to use the “boost” command to give certain keywords more importance-especially used in search engines:

Query to get results for Smith who work in California but the last name Smith takes more importance over California. Therefore, the output contains “Smith’s” who do not work in California as well.

```
# Boost results for Smith
GET bank/_search
{
  "query": {
    "bool": {
      "should": [
        { "match": { "state": "CA" } },
        { "match": { "last_name": "Smith", "boost": 10 } }
      ]
    }
  }
}
```

```
{
  "match": {
    "lastname": {
      "query": "Smith",
      "boost": 3
    }
  }
}
```

The screenshot shows the Elasticsearch Dev Tools interface with the 'Console' tab selected. The query entered is:

```
1 get bank/_search
2
3 get bank/_search
4 {
5   "query": {
6     "match": {
7       "state": "CA"
8     }
9   }
10 }
11
12
13
14
15 # Boost results for Smith
16 GET bank/_search
17 {
18   "query": {
19     "bool": {
20       "should": [
21         {
22           "match": {"state": "CA"} },
23           {
24             "lastname": {
25               "query": "Smith",
26               "boost": 3
27             }
28           }
29         ]
30     }
31   }
32 }
```

The response shows the total count and two hits, both of which are bank documents with state 'CA' and lastname 'Smith'. The first hit has a higher score due to the boost.

```

{
  "total": {
    "value": 18,
    "relation": "eq"
  },
  "max_score": 19.509869,
  "hits": [
    {
      "_index": "bank",
      "_id": "516",
      "_score": 19.509869,
      "_source": {
        "account_number": 516,
        "balance": 44940,
        "firstname": "Roy",
        "lastname": "Smith",
        "age": 37,
        "gender": "M",
        "address": "#0 Cherry Street",
        "email": "roysmith@parleynet.com",
        "city": "Carrsville",
        "state": "RI"
      }
    },
    {
      "_index": "bank",
      "_id": "68",
      "_score": 4.046554,
      "_source": {
        "account_number": 68,
        "balance": 44214,
        "firstname": "Hall",
        "lastname": "Key",
        "age": 25,
        "gender": "F",
        "address": "927 Bay Parkway",
        "email": "hallkey@eventex.com",
        "city": "Showmut",
        "state": "CA"
      }
    }
  ]
}
```

5. Writing a query using term

“Term” is used to look up numeric values and keywords

Text does not do well with term field as see below:

The screenshot shows the Elasticsearch Dev Tools interface. At the top, there's a navigation bar with the elastic logo, a search bar, and links for 'Setup guides', 'PC', and other tools like 'Search Profiler' and 'Grok Debugger'. Below that is a tab bar with 'Console' (which is selected), 'Search Profiler', 'Grok Debugger', and 'Painless Lab (BETA)'. A secondary navigation bar below the tabs includes 'History', 'Settings', 'Variables', and 'Help'. The main area is the 'Console' tab, which displays a code editor and a results panel. The code editor contains a GET request to 'bank/_search' with a query term for 'account_number': 516. The results panel shows a successful response (200 - OK) with a total time of 117 ms. The JSON response is as follows:

```
1 get bank/_search
2
3 GET bank/_search
4 {
5   "query": {"term": {
6     "account_number": 516
7   }}
8 }
```

```
2   "took": 1,
3   "timed_out": false,
4   "_shards": {
5     "total": 1,
6     "successful": 1,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": {
12      "value": 1,
13      "relation": "eq"
14    },
15    "max_score": 1,
16    "hits": [
17      {
18        "_index": "bank",
19        "_id": "516",
20        "_score": 1,
21        "_source": {
22          "account_number": 516,
23          "balance": 44940,
24          "firstname": "Roy",
25          "lastname": "Smith"
26          ,
27          "age": 37,
28          "gender": "M",
29          "address": "770
30          Cherry Street",
31          "
32        }
33      }
34    ]
35  }
```

While using term with text: This query returns no documents.

GET bank/_search

```
{
  "query": {"term": {
    "state": "RI"
  }}
}
```

The screenshot shows the Elasticsearch Dev Tools Console. The URL bar says "Setup guides". The navigation bar includes "Console", "Search Profiler", "Grok Debugger", and "Painless Lab (BETA)". The main area has tabs for "History", "Settings", "Variables", and "Help". A search bar at the top right contains the text "elastic". The code input field contains:

```
1 get bank/_search
2
3 GET bank/_search
4 {
5   "query": {"term": {
6     | "state": "RI"
7   }}
8 }
```

The response pane shows a successful 200-OK response with a duration of 117 ms. The JSON response is:

```
1 { "took": 1,
2   "timed_out": false,
3   "_shards": {
4     "total": 1,
5     "successful": 1,
6     "skipped": 0,
7     "failed": 0
8   },
9   "hits": {
10    "total": {
11      "value": 0,
12      "relation": "eq"
13    },
14    "max_score": null,
15    "hits": []
16  }
17 }
18 }
```

But, when match is used it shows 18 documents.

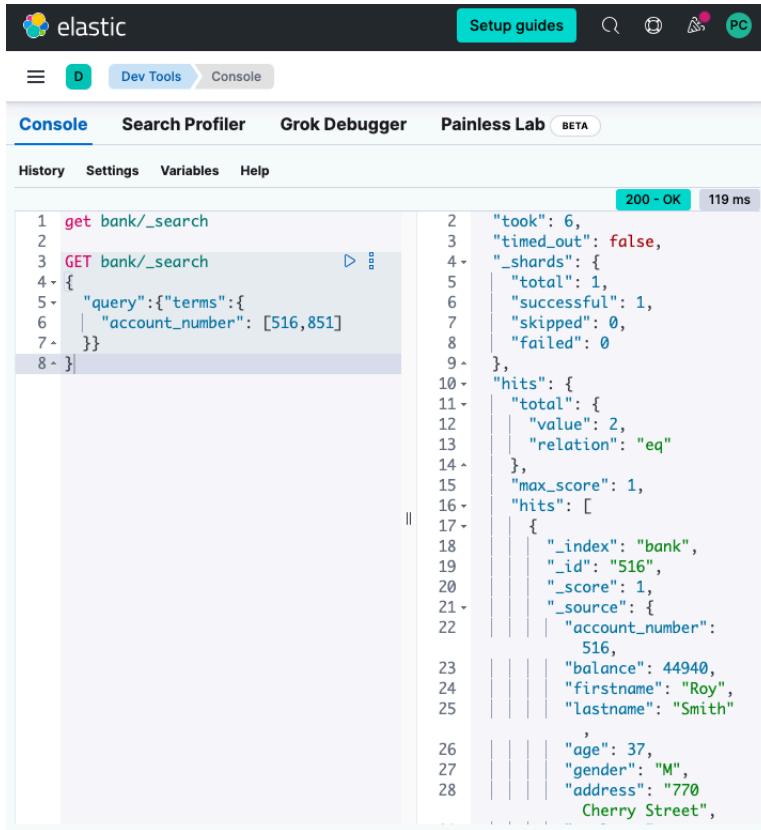
The screenshot shows the Elasticsearch Dev Tools Console. The URL bar says "Setup guides". The navigation bar includes "Console", "Search Profiler", "Grok Debugger", and "Painless Lab (BETA)". The main area has tabs for "History", "Settings", "Variables", and "Help". A search bar at the top right contains the text "elastic". The code input field contains:

```
1 get bank/_search
2
3 GET bank/_search
4 {
5   "query": {"match": {
6     | "state": "RI"
7   }}
8 }
```

The response pane shows a successful 200-OK response with a duration of 101 ms. The JSON response is:

```
1 { "took": 1,
2   "timed_out": false,
3   "_shards": {
4     "total": 1,
5     "successful": 1,
6     "skipped": 0,
7     "failed": 0
8   },
9   "hits": {
10    "total": {
11      "value": 18,
12      "relation": "eq"
13    },
14    "max_score": 3.990984,
15    "hits": [
16      {
17        "_index": "bank",
18        "_id": "49",
19        "_score": 3.990984,
20        "_source": {
21          "account_number": 49,
22          "balance": 29104,
23          "firstname": "Fulton",
24          "lastname": "Holt",
25          "age": 23,
26          "gender": "F",
27        }
28      }
29    ]
30  }
31 }
```

6. Query to retrieve documents using an array to get multiple results



The screenshot shows the Elasticsearch Dev Tools Console interface. At the top, there are tabs for 'Setup guides', 'Search Profiler', 'Grok Debugger', and 'Painless Lab (BETA)'. Below the tabs, there are buttons for 'History', 'Settings', 'Variables', and 'Help'. The main area displays a code snippet and its corresponding JSON response.

```
1 get bank/_search
2
3 GET bank/_search
4 {
5   "query": {"terms": {
6     "account_number": [516, 851]
7   }}
8 }
```

200 - OK 119 ms

```
2   "took": 6,
3   "timed_out": false,
4   "_shards": {
5     "total": 1,
6     "successful": 1,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": {
12      "value": 2,
13      "relation": "eq"
14    },
15    "max_score": 1,
16    "hits": [
17      {
18        "_index": "bank",
19        "_id": "516",
20        "_score": 1,
21        "_source": {
22          "account_number": 516,
23          "balance": 44940,
24          "firstname": "Roy",
25          "lastname": "Smith"
26          ,
27          "age": 37,
28          "gender": "M",
          "address": "770
          Cherry Street",
        }
      }
    ]
  }
}
```

7. Query using gte, gt, lte and lt -

This query searches for documents where the account number is in the range of 516 and 851.

Boost is used to increased the relevance score of the documents that match the condition by 2, which makes in relevant in the search results.

GET bank/_search

```
{
  "query": {
    "range": {
      "account_number": {
        "gte": 516,
        "lte": 851,
        "boost": 2
      }
    }
  }
}
```

8. Query to show all users above the age of 35

GET bank/_search

```
{  
  "query": {  
    "range": {  
      "age": {  
        "gt": 35  
      }  
    }  
  }  
}
```

The screenshot shows the Elasticsearch Dev Tools interface with the 'Console' tab selected. The search bar contains the query: `GET bank/_search`. The results pane displays the JSON response from the search. The response includes the '_source' field, which contains account details like balance, first name, last name, gender, address, and employer.

```
6 ## lt = Less-than  
7 # Show all accounts between  
8 # 516 and 851, boosting the  
# importance  
9 GET bank/_search  
10 {  
11   "query": {  
12     "range": {  
13       "account_number": {  
14         "gte": 516,  
15         "lte": 851,  
16         "boost": 2  
17       }  
18     }  
19   }  
20 }  
21 # Show all account holders  
22 # older than 35  
23 GET bank/_search >:  
24 {  
25   "query": {  
26     "range": {  
27       "age": {  
28         "gt": 35  
29       }  
30     }  
31   }  
32 }
```

```
1- [{"took": 4,  
"timed_out": false,  
"_shards": {  
  "total": 1,  
  "successful": 1,  
  "skipped": 0,  
  "failed": 0  
},  
"hits": {  
  "total": {  
    "value": 238,  
    "relation": "eq"  
  },  
  "max_score": 1,  
  "hits": [  
    {  
      "_index": "bank",  
      "_id": "6",  
      "_score": 1,  
      "_source": {  
        "account_number": 6,  
        "balance": 5686,  
        "firstname": "Hattie",  
        "lastname": "Bond",  
        "age": 36,  
        "gender": "M",  
        "address": "671 Bristol  
        Street",  
        "employer": "Netagy",  
        "middle_name": "Lillian",  
        "ssn": "123-45-6789",  
        "zip": "12345"  
      }  
    }  
  ]  
}
```

Tokenizer:

- Analyzes a query
- Handles a query

Here, I have used different tokenizers to see how it works;

When “standard” tokenizer is used for the text “The Moon is Made of Cheese Some Say”:

Every word is a token and is of the type “Alphanumeric”

GET bank/_analyze

```
{  
  "tokenizer": "standard",
```

```
"text" : "The Moon is Made of Cheese Some Say"
```

The screenshot shows the Elasticsearch Dev Tools interface. In the top navigation bar, there are tabs for 'Console', 'Search Profiler', 'Grok Debugger', and 'Painless Lab' (BETA). The 'Console' tab is selected. Below the tabs, there are sections for 'History', 'Settings', 'Variables', and 'Help'. The main area contains a code editor and a results panel.

Code Editor:

```
1 # Basic Example
2 GET bank/_analyze
3 {
4   "tokenizer" : "standard",
5   "text" : "The Moon is Made of Cheese
Some Say"
6 }

7
8 # Mixed String
9 GET bank/_analyze
10 {
11   "tokenizer" : "standard",
12   "text" : "The Moon-is-Made of Cheese
.Some Say$"
13 }

14
15 # Use the letter tokenizer
16 GET bank/_analyze
17 {
18   "tokenizer" : "letter",
19   "text" : "The Moon-is-Made of Cheese
.Some Say$"
20 }

21
22 # How about a URL
23 GET bank/_analyze
24 {
25   "tokenizer": "standard",
26   "text": "you@example.com login at
https://bensullins.com attempt"
27 }

28
29 GET bank/_analyze
30 {
31   "tokenizer": "uax_url_email",
32   "text": "you@example.com login at
https://bensullins.com attempt"
33 }

34
35 # Where it breaks, two fields with diff
analyzers
36 PUT /idx1
```

Results Panel:

```
1 {
2   "tokens": [
3     {
4       "token": "The",
5       "start_offset": 0,
6       "end_offset": 3,
7       "type": "<ALPHANUM>",
8       "position": 0
9     },
10    {
11      "token": "Moon",
12      "start_offset": 4,
13      "end_offset": 8,
14      "type": "<ALPHANUM>",
15      "position": 1
16    },
17    {
18      "token": "is",
19      "start_offset": 9,
20      "end_offset": 11,
21      "type": "<ALPHANUM>",
22      "position": 2
23    },
24    {
25      "token": "Made",
26      "start_offset": 12,
27      "end_offset": 16,
28      "type": "<ALPHANUM>",
29      "position": 3
30    },
31    {
32      "token": "of",
33      "start_offset": 17,
34      "end_offset": 19,
35      "type": "<ALPHANUM>",
36      "position": 4
37    },
38    {
39      "token": "Cheese",
40      "start_offset": 20,
41      "end_offset": 26,
42      "type": "<ALPHANUM>",

43  }
```

The results show the tokens extracted from the input text using the standard tokenizer. The word 'Say\$' is not tokenized as a single entity because the standard tokenizer does not consider punctuation as part of a word.

A “standard” tokenizer is used with the query “ The Moon-is-Made of Cheese.Some Say\$”

Every word is a token except ofr “Cheese.Some” is considered as one token and the “\$” is removed.

Type is Alphanumeric

```
# Mixed String
GET bank/_analyze
{
  "tokenizer" : "standard",
  "text" : "The Moon-is-Made of Cheese.Some Say$"
}
```

```

# Basic Example
GET bank/_analyze
{
  "tokenizer": "standard",
  "text": "The Moon is Made of Cheese
  Some Say"
}

# Mixed String
GET bank/_analyze
{
  "tokenizer": "standard",
  "text": "The Moon-is-Made of Cheese
  .Some Say$"
}

# Use the letter tokenizer
GET bank/_analyze
{
  "tokenizer": "letter",
  "text": "The Moon-is-Made of Cheese
  .Some Say$"
}

# How about a URL
GET bank/_analyze
{
  "tokenizer": "standard",
  "text": "you@example.com login at
  https://bensullins.com attempt"
}

GET bank/_analyze
{
  "tokenizer": "uax_url_email",
  "text": "you@example.com login at
  https://bensullins.com attempt"
}

# Where it breaks, two fields with diff
analyzers
PUT /idx1

```

The code above demonstrates various tokenizer behaviors. The first two examples use the standard tokenizer, which splits words by whitespace. The third example uses the letter tokenizer, which splits words by punctuation. The fourth example shows how the standard tokenizer handles URLs. The final example shows how two fields can have different analyzers.

The tokenizer “letter” is used with the text “The Moon-is-Made of Cheese.Some Say\$”.
 Every word is a token and the “\$” and “.” is removed.
 The type is “word”

To parse a url and the tokenizer used is “standard”
 Every word is a token and is of type alphanumeric
 GET bank/_analyze
 {
 "tokenizer": "standard",
 "text": "you@example.com login at https://bensullins.com attempt"
 }

```

1 # Basic Example
2 GET bank/_analyze
3 {
4   "tokenizer": "standard",
5   "text": "The Moon is Made of Cheese
    Some Say"
6 }
7
8 # Mixed String
9 GET bank/_analyze
10 {
11   "tokenizer": "standard",
12   "text": "The Moon-is-Made of Cheese
    .Some Say$"
13 }
14
15 # Use the letter tokenizer
16 GET bank/_analyze
17 {
18   "tokenizer": "letter",
19   "text": "The Moon-is-Made of Cheese
    .Some Say$"
20 }
21
22 # How about a URL
23 GET bank/_analyze
24 {
25   "tokenizer": "standard",
26   "text": "you@example.com login at
    https://bensullins.com attempt"
27 }
28
29 GET bank/_analyze
30 {
31   "tokenizer": "uax_url_email",
32   "text": "you@example.com login at
    https://bensullins.com attempt"
33 }
34
35 # Where it breaks, two fields with diff
36 analyzers
37
38 PUT /idx1

```

Click to send request

```

12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53

```

```

"start_offset": 4,
"end_offset": 15,
"type": "<ALPHANUM>",
"position": 1
},
{
"token": "login",
"start_offset": 16,
"end_offset": 21,
"type": "<ALPHANUM>",
"position": 2
},
{
"token": "at",
"start_offset": 22,
"end_offset": 24,
"type": "<ALPHANUM>",
"position": 3
},
{
"token": "https",
"start_offset": 25,
"end_offset": 30,
"type": "<ALPHANUM>",
"position": 4
},
{
"token": "bensullins.com",
"start_offset": 33,
"end_offset": 47,
"type": "<ALPHANUM>",
"position": 5
},
{
"token": "attempt",
"start_offset": 48,
"end_offset": 55,
"type": "<ALPHANUM>",
"position": 6
}
]

```

While parsing a url with the tokenizer “uax_url_email”

The email id is considered as a token and is of type “EMAIL”

URL is considered as type URL and the rest is alphanumeric

GET bank/_analyze

```

{
  "tokenizer": "uax_url_email",
  "text": "you@example.com login at https://bensullins.com attempt"
}
```

The screenshot shows the Elasticsearch Dev Tools interface. The top bar includes the elastic logo, a search bar, and buttons for 'Setup guides', 'PC', and 'PC' (likely a refresh or save icon). Below the header is a navigation bar with 'Console' (selected), 'Search Profiler', 'Grok Debugger', and 'Painless Lab (BETA)'. The main area has tabs for 'History', 'Settings', 'Variables', and 'Help'. The code editor contains a JSON script with line numbers from 24 to 62. The right panel displays the analysis results for the 'GET bank/_analyze' command, showing tokens like 'you@example.com', 'login', 'at', 'attempt', and URLs like 'https://bensullins.com'. The status bar at the bottom indicates a '200 - OK' response and '101 ms' execution time.

```

24+ {
25  "tokenizer": "standard",
26  "text": "you@example.com login at
27- https://bensullins.com attempt"
28-
29 GET bank/_analyze      D: [
30 { "tokenizer": "uax_url_email",
31   "text": "you@example.com login at
32     https://bensullins.com attempt"
33 }
34-
35 # Where it breaks, two fields with diff
36 analyzers
37 PUT /idx1
38- {
39   "mappings": {
40     "properties": {
41       "title": {
42         "type": "text",
43         "analyzer": "standard"
44       },
45       "english_title": {
46         "type": "text",
47         "analyzer": "english"
48     }
49   }
50- }
51-
52 GET idx1
53-
54 GET idx1/_analyze
55- {
56   "field": "title",
57   "text": "Bears"
58- }
59-
60 GET idx1/_analyze
61- {
62   "field": "english title".

```

```

1- [
2-   "tokens": [
3-     {
4-       "token": "you@example.com",
5-       "start_offset": 0,
6-       "end_offset": 15,
7-       "type": "<EMAIL>",
8-       "position": 0
9-     },
10-    {
11-      "token": "login",
12-      "start_offset": 16,
13-      "end_offset": 21,
14-      "type": "<ALPHANUM>",
15-      "position": 1
16-    },
17-    {
18-      "token": "at",
19-      "start_offset": 22,
20-      "end_offset": 24,
21-      "type": "<ALPHANUM>",
22-      "position": 2
23-    },
24-    {
25-      "token": "https://bensullins.com",
26-      "start_offset": 25,
27-      "end_offset": 47,
28-      "type": "<URL>",
29-      "position": 3
30-    },
31-    {
32-      "token": "attempt",
33-      "start_offset": 48,
34-      "end_offset": 55,
35-      "type": "<ALPHANUM>",
36-      "position": 4
37-    }
38-  ]
39- }

```

Creating a new index:

PUT /idx1

```
{
  "mappings": {
    "properties": {
      "title": {
        "type": "text",
        "analyzer": "standard"
      },
      "english_title": {
        "type": "text",
        "analyzer": "english"
      }
    }
  }
}
```

The screenshot shows the Elasticsearch Dev Tools Console interface. On the left, there is a code editor with a scroll bar. On the right, there is a preview pane showing the response to a POST request. A tooltip says "Click to send request".

```

24 - { "tokenizer": "standard",
25   "text": "you@example.com login at
26     https://bensullins.com attempt"
27 - }
28
29 GET bank/_analyze
30 {
31   "tokenizer": "uax_url_email",
32   "text": "you@example.com login at
33     https://bensullins.com attempt"
34
35 # Where it breaks, two fields with diff
36 analyzers
37 PUT /idx1
38 {
39   "mappings": {
40     "properties": {
41       "title": {
42         "type": "text",
43         "analyzer": "standard"
44       },
45       "english_title": {
46         "type": "text",
47         "analyzer": "english"
48     }
49   }
50 }
51
52 GET idx1/_analyze
53
54 GET idx1/_analyze
55 {
56   "field": "title",
57   "text": "Bears"
58 }
59
60 GET idx1/_analyze
61 {
62   "field": "english title".

```

```

1- {
2-   "idx1": {
3-     "aliases": {},
4-     "mappings": {
5-       "properties": {
6-         "english_title": {
7-           "type": "text",
8-           "analyzer": "english"
9-         },
10-        "title": {
11-          "type": "text",
12-          "analyzer": "standard"
13-        }
14-      },
15-      "settings": {
16-        "index": {
17-          "routing": {
18-            "allocation": {
19-              "include": {
20-                "_tier_preference": "data_content"
21-              }
22-            }
23-          }
24-        },
25-        "number_of_shards": "1",
26-        "provided_name": "idx1",
27-        "creation_date": "1720848374867",
28-        "number_of_replicas": "1",
29-        "uuid": "mk8zawhhSGyDxeIpBdiv2g",
30-        "version": {
31-          "created": "8505000"
32-        }
33-      }
34-    }
35-  }
36- }

```

Send data to the index to see the handling of data

GET idx1/_analyze

```
{
  "field": "title",
  "text": "Bears"
```

It is going to use standard analyzer and the result contains a token “bear” but the capital is dropped

The screenshot shows the Elasticsearch Dev Tools Console interface. At the top, there's a search bar with the placeholder "Find apps, content, and more." and a "Setup guides" button. Below the header, tabs for "Console", "Search Profiler", "Grok Debugger", and "Painless Lab (BETA)" are visible, with "Console" being the active tab. The main area is a code editor with syntax highlighting for JSON. The code consists of several lines of Elasticsearch requests and responses:

```
https://bensullins.com attempt"
27 }
28
29 GET bank/_analyze
30 {
31   "tokenizer": "uax_url_email",
32   "text": "you@example.com login at
https://bensullins.com attempt"
33 }
34
35 # Where it breaks, two fields with diff
analyzers
36 PUT /idx1
37 {
38   "mappings": {
39     "properties": {
40       "title": {
41         "type": "text",
42         "analyzer" : "standard"
43     },
44     "english_title": {
45       "type": "text",
46       "analyzer": "english"
47     }
48   }
49 }
50
51
52 GET idx1
53
54 GET idx1/_analyze
55 {
56   "field": "title",
57   "text": "Bears"
58 }
59
60 GET idx1/_analyze
61 {
62   "field": "english_title",
63   "text": "Bears"
64 }
65
```

The response on the right side shows the analyzed tokens for the "title" field. It includes a single token object with the following details:

```
1 {
2   "tokens": [
3     {
4       "token": "bears",
5       "start_offset": 0,
6       "end_offset": 5,
7       "type": "<ALPHANUM>",
8       "position": 0
9     }
10 ]
11 }
```

The "english_title" field is also listed below, indicating it was analyzed differently.

While using english_title it dropped the capital letter and the plural form so the token is “bear” and not Bears

GET idx1/_analyze

```
{
  "field": "english_title",
  "text": "Bears"
}
```

```

https://bensullins.com attempt
27-
28-
29 GET bank/_analyze
30- {
31   "tokenizer": "uox_url_email",
32   "text": "you@example.com login at
https://bensullins.com attempt"
33- }
34-
35 # Where it breaks, two fields with diff
      analyzers
36 PUT /idx1
37- {
38-   "mappings": {
39-     "properties": {
40-       "title": {
41-         "type": "text",
42-         "analyzer": "standard"
43-       },
44-       "english_title": {
45-         "type": "text",
46-         "analyzer": "english"
47-       }
48-     }
49-   }
50- }
51-
52 GET idx1
53-
54 GET idx1/_analyze
55- {
56   "field": "title",
57   "text": "Bears"
58- }
59-
60 GET idx1/_analyze
61- {
62   "field": "english_title",
63   "text": "Bears"
64- }
65

```

The screenshot shows the Elasticsearch Dev Tools Console interface. A POST request to `bank/_analyze` was made, resulting in a 200 OK response with a total execution time of 122 ms. The response body contains the analyzed tokens for the input text, specifically the word "bear". The tokens object is highlighted in the JSON response.

ANALYSIS: USING AGGREGATE FUNCTIONS

- See state count in aggregation as number of documents.

```

GET bank/_search
{
  "size":0,
  "aggs":{
    "states":{
      "terms":{"field":"state.keyword"}
    }
  }
}

```

```

1 GET bank/_search
2 {
3   "size":0,
4   "aggs":{
5     "states":{
6       "terms":{"field":"state.keyword"}
7     }
8   }
9 }

17  },
18  "aggregations": {
19    "states": {
20      "doc_count_error_upper_bound": 0,
21      "sum_other_doc_count": 743,
22      "buckets": [
23        {
24          "key": "TX",
25          "doc_count": 30
26        },
27        {
28          "key": "MD",
29          "doc_count": 28
30        },
31        {
32          "key": "ID",
33          "doc_count": 27
34        },
35        {
36          "key": "AL",
37          "doc_count": 25
38        },
39        {
40          "key": "ME",
41          "doc_count": 25
42        },
43        {
44          "key": "TN",
45          "doc_count": 25
46        },
47        {
48          "key": "WY",
49          "doc_count": 25
50        },
51        {
52          "key": "DC",
53          "doc_count": 24
54        },
55        {
56          "key": "MA",
57          "doc_count": 24
58        }
59      ]
60    }
61  }
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

2. Average Balance using nested aggregations - Query to get average balance of accounts in states and also mention the number of documents in each state.

GET bank/_search

```
{
  "size":0,
  "aggs":{
    "states":{
      "terms":{"field":"state.keyword"},

      "aggs":{
        "avg_bal":{
```

```

        "avg": {
          "field": "balance"
        }
      }
    }
  }
}

```

The screenshot shows the Elasticsearch Dev Tools Console interface. At the top, there's a navigation bar with tabs for 'Dev Tools' (which is active), 'Search Profiler', 'Grok Debugger', and 'Painless Lab (BETA)'. Below the tabs is a sub-navigation bar with 'History', 'Settings', 'Variables', and 'Help'.

The main area displays a search request (line 11) and its response (lines 1-42). The request is a GET request to 'bank/_search' with a size of 0 and an aggregation named 'avg_bal' which uses the 'balance' field. The response shows the execution took 7ms, had 1 shard, and 1 total document. It includes a 'hits' section with a total value of 1000 and a relation of 'eq'. The 'aggregations' section shows three buckets for states: TX, MD, and ID. The TX bucket has 30 documents and an average balance of 26073.3. The MD bucket has 28 documents and an average balance of 26161.535714285714. The ID bucket has 27 documents and an average balance of 24368.777777777777.

```

1 GET bank/_search
2 {
3   "size":0,
4   "aggs":{
5     "states":{
6       "terms":{"field": "state.keyword"}
7     }
8   }
9 }
10
11 GET bank/_search
12 {
13   "size":0,
14   "aggs":{
15     "states":{
16       "terms":{"field": "state.keyword"}, "aggs":{ "avg_bal":{ "avg":{ "field": "balance" } } }
17     }
18   }
19 }
20
21
22
23
24 }
25
26 }

1 [
2   "took": 7,
3   "timed_out": false,
4   "_shards": {
5     "total": 1,
6     "successful": 1,
7     "skipped": 0,
8     "failed": 0
9   },
10   "hits": {
11     "total": {
12       "value": 1000,
13       "relation": "eq"
14     },
15     "max_score": null,
16     "hits": []
17   },
18   "aggregations": {
19     "states": {
20       "doc_count_error_upper_bound": 0,
21       "sum_other_doc_count": 743,
22       "buckets": [
23         {
24           "key": "TX",
25           "doc_count": 30,
26           "avg_bal": {
27             "value": 26073.3
28           }
29         },
30         {
31           "key": "MD",
32           "doc_count": 28,
33           "avg_bal": {
34             "value": 26161.535714285714
35           }
36         },
37         {
38           "key": "ID",
39           "doc_count": 27,
40           "avg_bal": {
41             "value": 24368.777777777777
42           }
43         }
44       ]
45     }
46   }
47 }

```

3. Add a further aggregation to get the number of accounts in each state, avg account balance and for every age- how many documents exist?

GET bank/_search

```
{
  "size":0,
  "aggs":{
    "states":{

```



```

"terms": {"field": "state.keyword"},  

"aggs": {  

    "avg_bal": {  

        "avg": {  

            "field": "balance"  

        }  

    },  

    "age": {  

        "terms": {  

            "field": "age"  

        },  

        "aggs": {"avg_bal": {"avg": {"field": "balance"} } }  

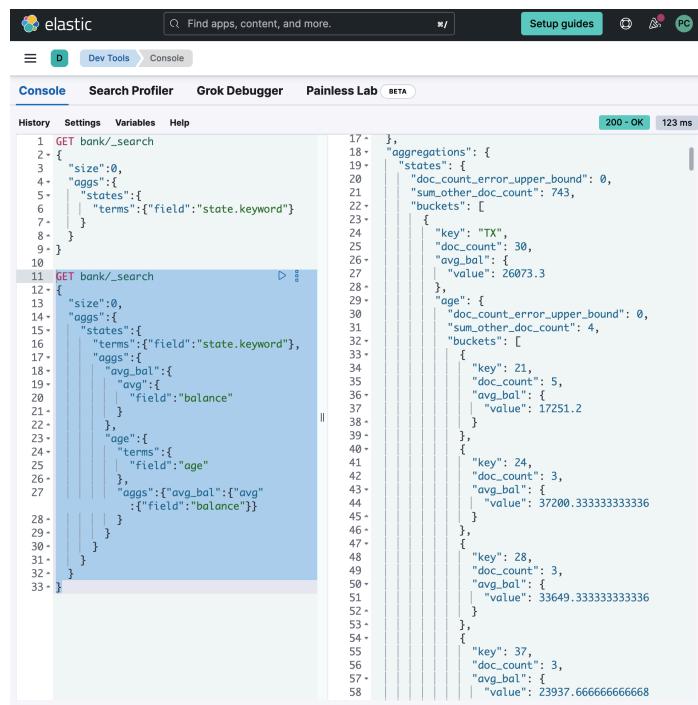
    }  

}  

}  

}

```



The screenshot shows the Elasticsearch Dev Tools interface with the 'Console' tab selected. The history pane displays two requests:

- Request 1:** A GET request to `bank/_search` with size 0 and no aggs. The response shows aggregations for states, with a total of 743 documents and an average balance of 26073.3.
- Request 2:** A GET request to `bank/_search` with size 0 and aggs. The response shows aggregations for states and age. For state "TX", there are 30 documents with an average balance of 17251.2. For state "CA", there are 3 documents with an average balance of 37200.33333333336. For state "NY", there are 3 documents with an average balance of 33649.33333333336. For state "IL", there are 3 documents with an average balance of 23937.666666666668.

```

History Settings Variables Help 200 - OK | 123 ms
1 GET bank/_search
2 {
3   "size":0,
4   "aggs":{
5     "states":{
6       "terms":{ "field": "state.keyword" }
7     }
8   }
9 }
10
11 GET bank/_search
12 {
13   "size":0,
14   "aggs":{
15     "states":{
16       "terms":{ "field": "state.keyword" },
17       "aggs":{ "avg":{ "field": "balance" } }
18     },
19     "age":{ "terms":{ "field": "age" },
20       "aggs":{ "avg_bal":{ "avg":{ "field": "balance" } } } }
21   },
22   "aggs":{ "terms":{ "field": "age" },
23     "aggs":{ "avg":{ "field": "balance" } } }
24 }
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58

```

3. Get stats on the dataset present in the cluster

GET `bank/_search`

```
{
  "size":0,
  "aggs":{
```

```
"balance-stats":{  
  "stats":{  
    "field":"balance"  
  }  
}  
}  
}  
}
```

The screenshot shows the Elasticsearch Dev Tools interface with the 'Console' tab selected. At the top, there's a search bar with placeholder text 'Find apps, content, and more.' and a 'Setup guides' button. Below the header, tabs for 'Console', 'Search Profiler', 'Grok Debugger', and 'Painless Lab (BETA)' are visible.

The main area displays a search request and its response. On the left, the request is shown as a code block:

```
1 GET bank/_search  
2 {  
3   "size":0,  
4   "aggs":{  
5     "balance-stats":{  
6       "stats":{  
7         "field":"balance"  
8       }  
9     }  
10    }  
11  }  
12
```

On the right, the response is displayed as a JSON object with line numbers:

```
1 {  
2   "took": 5,  
3   "timed_out": false,  
4   "_shards": {  
5     "total": 1,  
6     "successful": 1,  
7     "skipped": 0,  
8     "failed": 0  
9   },  
10  "hits": {  
11    "total": {  
12      "value": 1000,  
13      "relation": "eq"  
14    },  
15    "max_score": null,  
16    "hits": []  
17  },  
18  "aggregations": {  
19    "balance-stats": {  
20      "count": 1000,  
21      "min": 1011,  
22      "max": 49989,  
23      "avg": 25714.837,  
24      "sum": 25714837  
25    }  
26  }  
27 }
```

The response indicates a successful search with a took time of 5ms, 1000 hits, and various aggregation statistics for the 'balance' field.

4.What number of accounts are percentiles? This query gives us the distribution

```
{  
  "size":0,  
  "aggs":{  
    "pct_balance":{  
      "percentiles":{  
        "field":"balance",  
        "percents":[  
          1,  
          5,  
          25,  
          50,  
          75,  
          95,  
          99  
        ]  
      }  
    }  
  }  
}
```

The screenshot shows the Elasticsearch Dev Tools Console interface. At the top, there are tabs for History, Settings, Variables, Help, and a selected Dev Tools tab. Below the tabs, the URL is /_search and the status is 200 - OK. The response time is 122 ms. The main area displays the search query and its results. The query is:

```
1 GET bank/_search  
2 {  
3   "size":0,  
4   "aggs":{  
5     "pct_balance":{  
6       "percentiles":{  
7         "field":"balance",  
8         "percents":[  
9           1,  
10          5,  
11          25,  
12          50,  
13          75,  
14          95,  
15          99  
16        ]  
17      }  
18    }  
19  }  
20 }  
21 }
```

The results are:

```
1 {  
2   "took": 14,  
3   "timed_out": false,  
4   "_shards": {  
5     "total": 1,  
6     "successful": 1,  
7     "skipped": 0,  
8     "failed": 0  
9   },  
10  "hits": {  
11    "total": {  
12      "value": 1000,  
13      "relation": "eq"  
14    },  
15    "max_score": null,  
16    "hits": []  
17  },  
18  "aggregations": {  
19    "pct_balance": {  
20      "values": {  
21        "1.0": 1462.84,  
22        "5.0": 1535.85,  
23        "25.0": 13705.75,  
24        "50.0": 26033.5,  
25        "75.0": 38179.25,  
26        "95.0": 47551.549999999996,  
27        "99.0": 49339.16  
28      }  
29    }  
30  }  
31 }
```

4. What percentile of accounts contain a balance of 35,000 to 50,000. See if there are any outliers present and can be used for fraud detection .

Accounts with 35,000 is in 68 percentile and 50,000 are in the 100 percentile

GET bank/_search

```
{
  "size":0,
  "aggs":{
    "bal_outlier":{
      "percentile_ranks":{
        "field":"balance",
        "values":[35000,50000]
      }
    }
  }
}
```

The screenshot shows the Elasticsearch Dev Tools interface with the 'Console' tab selected. The query entered is:

```
GET bank/_search
{
  "size":0,
  "aggs":{
    "bal_outlier":{
      "percentile_ranks":{
        "field":"balance",
        "values":[35000,50000]
      }
    }
  }
}
```

The response is displayed in the right panel:

```
{
  "took": 5,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 1000,
      "relation": "eq"
    },
    "max_score": null,
    "hits": []
  },
  "aggregations": {
    "bal_outlier": {
      "values": [
        "35000.0": 68.99162303664922,
        "50000.0": 100
      ]
    }
  }
}
```

The status bar at the top indicates a 200 - OK response with a duration of 117 ms.

5. This query shows the distribution of accounts and its balance

GET bank/_search

```
{
  "size":0,
  "aggs":{
    "bals":{
      "histogram":{
        "field":"balance",
        "interval":1000
      }
    }
  }
}
```

```
"interval":500
}

}
}

}

1 GET bank/_search
2 {
3   "size":0,
4   "aggs":{
5     "bals":{
6       "histogram":{
7         "field":"balance",
8         "interval":500
9       }
10      }
11    }
12  }
13 }
```

The screenshot shows the Elasticsearch Dev Tools interface with the 'Console' tab selected. A search bar at the top contains the query 'Find apps, content, and more.' and a URL field with '/'. To the right are 'Setup guides', a gear icon, a person icon, and a 'PC' icon.

The main area displays a search result for the '_search' endpoint. The status bar indicates '200 - OK' and '104 ms'.

The search results show a histogram aggregation for the 'balance' field. The output is a JSON object with the following structure:

```
doc_count": 8
},
{
  "key": 6000,
  "doc_count": 16
},
{
  "key": 6500,
  "doc_count": 6
},
{
  "key": 7000,
  "doc_count": 6
},
{
  "key": 7500,
  "doc_count": 8
},
{
  "key": 8000,
  "doc_count": 8
},
{
  "key": 8500,
  "doc_count": 14
},
{
  "key": 9000,
  "doc_count": 5
},
{
  "key": 9500,
  "doc_count": 10
},
{
  "key": 10000,
  "doc_count": 11
},
{
  "key": 10500,
  "doc_count": 15
},
```