Elasticsearch Queries

(A)Full text queries:-

* The full text queries enables you to search analyzed text fields such as body of an email. The query string is processed using the same analyzer that was applied to the field during indexing.

1. Intervals Query:-

* The intervals query uses matching rules, constructed from a small set of definitions. These rules are then applied to terms from a specified field.
* The following intervals search returns document containing ‘my favourite food’ immediately followed by ‘hot water’ or ‘cold porridge’ in my\_text field.

{

"query": {

"intervals" : {

"my\_text" : {

"all\_of" : { //means all must be true

"ordered" : true,

"intervals" : [

{

"match" : {

"query" : "my favorite food",

"max\_gaps" : 0,

"ordered" : true //matching term must appear

} in their specified order

},

{

"any\_of" : {

"intervals" : [

{ "match" : { "query" : "hot water" } },

{ "match" : { "query" : "cold porridge" } }

]

}

}

]

}

}

}

}

}

* Filter rule returns intervals based on a query
* It returns the documents that have a words hot and porridge within 10 positions of each other, without the word salty in between.

{

"query": {

"intervals" : {

"my\_text" : {

"match" : {

"query" : "hot porridge",

"max\_gaps" : 10,

"filter" : {

"not\_containing" : {

"match" : {

"query" : "salty"

}

}

}

}

}

}

}

}

* after, before, contained\_by, containing, not\_contained\_by, not\_containing, not\_overlapping, overlapping, script

(2) match query:-

* Returns documents that match a provided text, number, or Boolean value.
* The match query is standard query for performing a full text search, including options for fuzzy matching.

{

"query": {

"match" : {

"message" : {

"query" : "this is a test"

}

}

}

}

(3) match Boolean prefix query:-

* This query analyzes its input and constructs a bool query from terms. Each term except the last is used in term query. The last term is used prefix query.

{

"query": {

"match\_bool\_prefix" : {

"message" : "quick brown f"

}

}

}

* Analysis produces the term quick, brown, and f is similar to the following bool query.

{

"query": {

"bool" : {

"should": [

{ "term": { "message": "quick" }},

{ "term": { "message": "brown" }},

{ "prefix": { "message": "f"}}

]

}

}

}

(4) match phrase query:-

* This query requires that all the terms in the query string be present in the document, be in the order specified in the query string and be close to each other. You can specify slop value which indicates how far apart terms are allowed to be while still considering the document a match.

{

"query": {

"match\_phrase" : {

"message" : "this is a test"

}

}

}

(5) match phrase prefix query:-

* Returns document that contain the words of a provided text, in the same order as provided. The last term of the provided text is treated as prefix, matching any words that begin with that term.

{

"query": {

"match\_phrase\_prefix" : {

"message" : {

"query" : "quick brown f"

}

}

}

}

(6) multi match query:-

* This query builds on the match query to allow multi fields queries.

{

"query": {

"multi\_match" : {

"query": "this is a test",

"fields": [ "subject", "message" ]

}

}

}

* Fields can be specified with the wildcards.

{

"query": {

"multi\_match" : {

"query": "Will Smith",

"fields": [ "title", "\*\_name" ]

}

}

}

* Query the title, first\_name, and last\_name fields.
* Individual fields can be boosted with caret notation.

{

"query": {

"multi\_match" : {

"query" : "this is a test",

"fields" : [ "subject^3", "message" ]

}

}

}

* The subject filed is three times as important as the message field.

(7) common terms query:-

* This query is modern alternative to stopwords which improves the precision and recall of search results, without sacrificing performance.
* The common term query divides the query terms into two groups: more important and less important.

{

"query": {

"common": {

"body": {

"query": "nelly the elephant as a cartoon",

"cutoff\_frequency": 0.001,

"low\_freq\_operator": "and"

}

}

}

}

{

"query": {

"bool": {

"must": [

{ "term": { "body": "nelly"}},

{ "term": { "body": "elephant"}},

{ "term": { "body": "cartoon"}}

],

"should": [

{ "term": { "body": "the"}},

{ "term": { "body": "as"}},

{ "term": { "body": "a"}}

]

}

}

}

(8) Query string query:-

* This query uses a syntax to sparse and split the provided query string based on operators such as AND or NOT. The query then analyzes each split text independently before returning matching documents. It return error for any invalid syntax.

{

"query": {

"query\_string" : {

"query" : "(new york city) OR (big apple)",

"default\_field" : "content"

}

}

}

(9) simple query string query:-

* This query is same as previous one but does not return error for invalid syntax.

{

"query": {

"simple\_query\_string" : {

"query": "\"fried eggs\" +(eggplant | potato) -frittata",

"fields": ["title^5", "body"],

"default\_operator": "and"

}

}

}

(B)Compound Queries:-

* Compound queries wrap other leaf queries, either to combine their results and scores.

(1) bool query:-

* It is built using one or more Boolean clauses , each clause with typed occurrence. [occurrence = must, must\_not, should, filter]
* Example:

{

"query": {

"bool" : {

"must" : {

"term" : { "user" : "kimchy" }

},

"filter": {

"term" : { "tag" : "tech" }

},

"must\_not" : {

"range" : {

"age" : { "gte" : 10, "lte" : 20 }

}

},

"should" : [

{ "term" : { "tag" : "wow" } },

{ "term" : { "tag" : "elasticsearch" } }

],

"minimum\_should\_match" : 1,

"boost" : 1.0

}

}

}

* Bool query takes a more matches is better approach, so the score from each matching ‘must’ or ‘should’ clause will be added to score for each document.

1. boosting query:-
   * Returns documents matching a positive query while reducing the relevance score of documents that also match a negative query.
   * You can use the boosting query to demote certain documents without excluding them from the search results.

{

"query": {

"boosting" : {

"positive" : {

"term" : {

"text" : "apple"

}

},

"negative" : {

"term" : {

"text" : "pie tart fruit crumble tree"

}

},

"negative\_boost" : 0.5

}

}

}

* If a returned document matches the positive query and this query, the boosting query calculates the final relevance score for the document as follows:

1. Take the original relevance score from positive query.
2. Multiply the score by the negative\_boost value.

(3) constant score query:-

* Wraps a filter query and returns every matching document with a relevance score equal to the boost parameter value.

{

"query": {

"constant\_score" : {

"filter" : {

"term" : { "user" : "kimchy"}

},

"boost" : 1.2

}

}

}

(4) Disjunction match query:-

* If a returned document matches multiple query clauses, the dis\_max query assigns the document the highest relevance score from any matching clause, plus a tie breaking increment for any additional matching subqueries.

{

"query": {

"dis\_max" : {

"queries" : [

{ "term" : { "title" : "Quick pets" }},

{ "term" : { "body" : "Quick pets" }}

],

"tie\_breaker" : 0.7

}

}

}

* tie\_breaker is a floating point number from 0.0 to 1.0 used to increase the relevance score of the documents matching multiple query clauses.
* If a document matches multiple clauses, the dis\_max query calculates the relevance score for document as follows:

1. Take the relevance score from matching clause with the highest score.
2. Multiply the score from any other matching clauses by the tie\_breaker value.
3. Add the highest score to multiplied scores.

(5) Function score query:-

* The function\_score allows you to modify the score of documents that are retrieved by a query.
* This can be useful if, for example, a score function is computationally expensive and it is sufficient to compute the score on a filtered set of documents.
* To use a function\_score, a user has to define a query and one or more functions, that compute a new score for each document returned by a query.

{

"query": {

"function\_score": {

"query": { "match\_all": {} },

"boost": "5",

"random\_score": {},

"boost\_mode":"multiply"

}

}

}