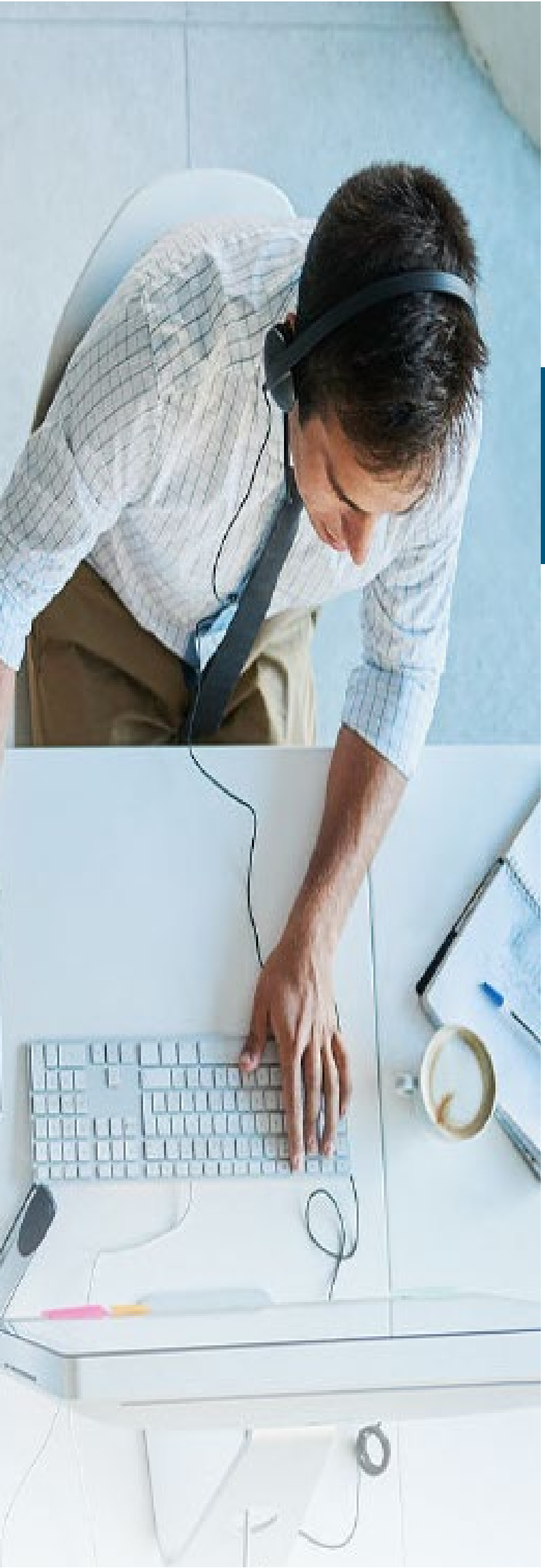

CALL RECORDING

Moments Query Language

Admin Quick Reference

MOMENTUM

Powered By: MiaRec

UC24Q2

Table of contents

Introduction

Moments Query Language (MQL) is a special-purpose query language designed to analyze call transcripts. The expressions written in MQL are used in the Call Recording platform for the following tasks:

- extracting topics and keywords from a transcript
- automatic evaluation of agents
- redacting sensitive data from recordings and transcripts, for example, credit card numbers

Advantages of MQL

MQL lets you search not only the words spoken, but also temporal patterns like beginning/end of a call, nearness, speaker (agent vs customer), fuzzy matching and more.

Examples of such queries:

```
error NEAR:5 website
```

Search for the word "**error**" spoken near the word "**website**", with a distance between these words no more than 5 words.

```
AGENT: ("my name is" OR "this is")
```

Search for an agent to say the words "**my name is**" or "**this is**"

```
R"[0-9]+" AFTER ("credit card" OR "security code")
```

Search for any digits (0-9) spoken after the anchor words "**credit card**" or "**security code**"

```
POSAFTER:-50 CUSTOMER: thank*
```

Search in the last 50 words of the conversation (i.e. the end of a call) for a customer saying words that match a wildcard pattern `thank*` (i.e. match "thank", "thanks", etc)

MQL lets you query the metadata attributes like call duration, sentiment score, etc.

Examples of such queries:

```
AGENT: (problem NEAR ship*) AND $call-duration > "2:00"
```

Search for an agent to say the word "**problem**" near a word that starts with `ship` (e.g. "**shipment**", "**shipping**", as well as probably non-relevant "**ship**"), but only when a call duration is longer than 2 minutes.

```
(cancel NEAR order) AND $sentiment-score < -50
```

Search for the word "**cancel**" near the word "**order**", but only when a sentiment score is lower than -50 (very negative).

More examples of MQL expressions

```
error
```

Search for the word "**error**" in a transcript

```
error NEAR:5 website
```

Search for the word **"error"** spoken near the word **"website"**, with a distance between these two words no more than 5 words. Such a query would match phrases like **"error on your website"**, **"website shows an error"**, but not **"error when I tried to order the service on your website"**, because the distance between the searched words in this last example is more than 5.

```
("speak to" OR "transfer to") ONEAR (supervisor OR manager)
```

Search for the phrase that begins with the anchor words **"speak to"** or **"transfer to"** followed by either the word **"supervisor"** or **"manager"**, i.e. such an expression will match phrases like **"speak to supervisor"**, **"speak to manager"**, **"transer to supervisor"**, as well as **"speak to YOUR supervisor"** (i.e. with other words in a middle).

```
AGENT: "my name is"
```

Search for the phrase **"my name is"** spoken by an agent.

```
 POSBEFORE:50 AGENT: ("my name is" OR "this is")
```

Search for the phrase **"my name is"** or **"this is"** spoken by an agent in the first 50 words of a conversation.

```
 POSAFTER:-20 CUSTOMER: thank*
```

Search for any word that starts with **"thank"** (e.g. *"thank"*, *"thanks"*) spoken by a customer in the last 20 words of the conversation.

```
R"[0-9]+" AFTER:50 ("credit card" OR "security code")
```

Search for any digit (0-9) spoken after the anchor words **"credit card"** or **"security code"**, with a distance of up to 50 words between the anchor words and the digits.

MQL Syntax

Searching for Phrases and Words

MQL supports searching for a phrase or a word using the following text matchers:

Matcher	Example	Description
Word	`shipment`	Search for the word "**shipment**" in a transcript
Quoted Term	`"problem with a shipment"`	Search for the exact phrase "**problem with a shipment**" in a transcript
Simple (wildcard) pattern	`ship*`	Search for the words that begin with "**ship**", like "**shipment**", "**shipping**", as well as "**ship**"
Regex pattern	`R"ship(ment|ping)"`	Search for the words matching the regular expression, in this example, it matches the words "**shipping**" and "**shipment**". Note, with such a regular expression, **ship** doesn't have to be at the beginning of the word, i.e. it will match the word **pre-shipment** as well. To enforce a match on word boundaries, add `\b` (boundary of word) to the regular expression, for example, `R"\bship(ment|ping)\b"`.

Word and Quoted Term matches

A Quoted Term matcher requires that the text matches literally to the search term. This is best demonstrated in the following examples.

```
"cancel order"
```

This Quoted Term expression matches the phrase **"cancel order"**, but not **"cancel my order"**. To overcome such a limit, you can use the operator `OR` to list all the variants of a phrase, like:

```
"cancel order" OR "cancel my order" OR "cancel this order"
```

```
cancel ONEAR:5 order
```

This expression consists of two Word matchers (the words **cancel** and **order**), with a proximity operator **ONEAR:5** between them.

The `ONEAR:5` operator instructs the search engine to find the word **"cancel"** followed by the word **"order"**, with a distance between these two words no more than 5.

Such an expression matches phrases "cancel order", "cancel my order", "cancel my recent order", etc.

Note 1. The operator `ONEAR` is order-dependent, i.e. the word **"cancel"** must appear in a transcript before the word **"order"**. There is an alternative operator `NEAR` that is order-independent.

Note 2. The operator `ONEAR:5` can be omitted because it is a default operator in MQL expressions, i.e. the expression `cancel order` is the same as `cancel ONEAR:5 order`.

```
cancel NEAR:5 order
```

This expression uses an order-independent operator **NEAR**, which instructs the search engine to find the words **"cancel"** and **"order"**, that appear in a transcript close to each other (distance up to 5 words), the order of appearance of the searched words is not important.

Such an expression matches **"cancel my order"** as well as **"order that I want to cancel"**, where words appear in reverse order.

Case insensitiveness

Both Word and Quoted Term matchers are case-insensitive, e.g. the expression `order` will match **"order"**, **"Order"**, **"ORDER"** and "oRDER".

Escaping a quote character

To search for a quote symbol (`"`) literally, repeat it twice `""` . Example:

```
"foo "" bar"
```

This expression matches the phrase **foo " bar**.

Note, the double `""` is supported in a Quoted Term only. It is a syntax error to use a quote inside a Word matcher, like `foo""bar` , but it is ok to use it in a Quoted Term matcher, like `"foo""bar"` .

Simple (wildcard) pattern

A Word matcher supports wildcard pattern matching.

The following table describes wildcard patterns, listing the pattern and its use.

Pattern	Use	Example
`*`	Match zero or more characters	`bl*` matches **bl**, **black**, **blue**, and **blob**
`?`	Match exactly one occurrance of any character	`h?t` finds **hot**, **hat**, and **hit**
`[abc]`	Match one occurance of the characters **a**, **b**, or **c**.	`h[oa]t` finds **hot** and **hat**, but not **hit**
`[!az]`	Match any characters except **a** or **z**	`h[!oa]t` finds **hit**, but not **hot** and **hat**
`[a-c]`	Match one occurance of a character between **a** and **c**	`c[a-c]t` finds **cat** and **cbt**, but not **cut**

> **Note**
>
> The wildcard patterns are supported in a Word matcher only. A Quoted Term interprets those symbols literally. For example `bl*` is a pattern match, but `"bl*"` is the exact match.
>
> Such a difference between Word and Quoted Term matchers is useful when you need to search for one of the wildcard symbols literally in a text.
>
> For example, to find an exclamation point in a text, use a Quoted Term expression, like `"Great!"`

Regular expression (REGEX) pattern

To match complex text patterns, use Regular expressions (REGEX).

The regular expression must be enclosed into `R"` and `"` characters. Examples:

Pattern	Use
`R"[0-9]+`	Match one of more digits in a text
`R"ship(ment|ping)`	Match words "**shipment**" and "**shipping**"

To match a quote (`"`) character in a regular expression, include it twice, like `R"foo""bar"` . MiaRec supports standard regular expression patterns.

A regular expression may use any of the following metacharacters:

`.`

Matches any single character. For example:

```
a.c
```

... will match "abc", but not "ac" or "abbc"

`[]`

A bracket expression. Matches a single character that is contained within the brackets. For example:

```
[abc]
```

... will match "a", "b" or "c".

```
[hc]at
```

... will match "hat" and "cat".

A `-` character between two other characters forms a range that matches all characters from the first character to the second. For example:

```
[0-9]
```

... will match any decimal digit.

```
[a-z]
```

... will match any lowercase letter from "a" to "z". These forms can be mixed:

```
[abcx-z]
```

... will match "a", "b", "c", "x", "y" or "z".

To include a literal `-` character, it must be written first or last, for example, `[abc-]`, `[-abc]`.

To include a literal `]` character, it must immediately follow the opening bracket `[`, for example, `[]abc]`.

`[^ ]`

Matches a single character that is not contained within the brackets. For example:

```
[^abc]
```

... will match any character other than "a", "b", or "c".

```
[^a-z]
```

... will match any single character that is not a lowercase letter from "a" to "z". As above, literal characters and ranges can be mixed, like `[^abcx-z]`

`*`

Matches the preceding element zero or more times. For example:

```
a*c
```

... will match "ac", "abc", "abbbc" etc.

```
[0-9]*
```

... will match "" (empty string), "0", "1", "2", "14", "502", "98541654", and so on (any combination of digits).

`( )*`

Matches zero of more instances of the characters sequence, specified inside parentheses. For example:

```
(ab)*
```

... will match "", "ab", "abab", "ababab", and so on.

```
(1234)*
```

... will match "", "1234", "12341234", "123412341234", and so on.

`+`

Matches the preceding element one or more times. For example:

```
ba+
```

... will match "ba", "baa", "baaa", and so on.

```
0[0-9]+
```

... will match "00", "01", "02", "001", "01234", "09876543210", or any other combination of digits with preceding 0 and minimum length equal to two characters.

`?`

Matches the preceding element zero or one time. For example:

```
ba?
```

... will match "b", or "ba", but not "baa"

```
0[0-9]?
```

... will match "0", "01", "02", "03", and so on.

`|`

The choice (aka alternation or set union) operator matches either the expression before or the expression after the operator. For example:

```
abc|def
```

... will match "abc" or "def".

```
(0|011)[1-9]+
```

... will match phone number, which starts with either 0 or 011.

`{n}`

Matches the preceding element exactly n times. For example:

```
a{3}
```

... will match "aaa", but not "a", "aa" or "aaaa"

```
[0-9]{5}
```

... will match "01234", "56789" or any other combination of digits, which has lenght 5 characters.

`{m, n}`

Matches the preceding element at least m and not more than n times. For example:

```
a{3,5}
```

... will match "aaa", "aaaa", "aaaaa", but not "aa" or "aaaaaaaa".

`{m, }`

Matches the preceding element at least m times. For example:

```
a{2,}
```

... will match "aa", "aaa", "aaaa", and so on.

`^`

Matches the beginning of a string. For example:

```
^[hc]at
```

... will match "hat" and "cat", but only at the beginning of the string

`$`

Matches the end of a string. For example:

```
[hc]at$
```

... will match "hat" and "cat", but only at the end of the string

```
^[hc]at$
```

... will match "hat" and "cat", but only when the string contains no other characters

`\`

Backslash (`\`) character is used for escaping metacharacters. For example:

```
1+2
```

... will match "12", "112", "11112", but not "1+2", because "plus" character has a special meaning (see above).

```
1\+2
```

... will match exactly "1+2". In this example, "plus" character is escaped with backslash character (`\+`).

Searching for Named Entities

Named-entity recognition (NER) is a subtask of information extraction that seeks to locate and classify named entities mentioned in unstructured text into pre-defined categories such as person names, organizations, locations, time expressions, quantities, monetary values, etc.

MiaRec voice analytics automatically extract the following names entity classes from a transcript: Table 1. Supported named entity classes

Named entity class	Description
`#CARDINAL`	Numerals that do not fall under another type
`#DATE`	Absolute or relative dates or periods
`#EVENT`	Named hurricanes, battles, wars, sports events, etc.
`#FAC`	Buildings, airports, highways, bridges, etc.
`#GPE`	Countries, cities, states
`#LANGUAGE`	Any named language
`#LAW`	Named documents made into laws.
`#LOC`	Non-GPE locations, mountain ranges, bodies of water
`#MONEY`	Monetary values, including unit
`#NORP`	Nationalities or religious or political groups
`#ORDINAL`	"first", "second", etc.
`#ORG`	Companies, agencies, institutions, etc.
`#PERCENT`	Percentage, including "%"
`#PERSON`	People, including fictional
`#PRODUCT`	Objects, vehicles, foods, etc. (not services)
`#QUANTITY`	Measurements, as of weight or distance
`#TIME`	Times smaller than a day
`#WORK_OF_ART`	Titles of books, songs, etc.

Using NER classes in MQL expressions

Named entity classes can be included in MQL expression.

For example, the class `#PERSON` can be used in data redaction expression to automatically remove person names from audio recordings and transcript.

Another sample expression

```
R"[0-9]+" NOTIN #MONEY
```

In the above example, we are searching for digits 0 to 9 (using the Regex pattern `[0-9]+`), but not if they are found inside a text labeled with MONEY class.

Searching by Metadata

You can use call attributes in MQL expressions.

For example, if you would like to search for a greeting phrase "Thank you for calling" for inbound calls only, then the expression can include `$direction` attribute, like in:

```
AGENT: "thank you for calling" AND $direction = inbound
```

Call attributes

Table 1. Support call attributes

Attribute	Type	Description	Example
`$caller-number`	text	Caller party phone number	`$caller-number = "12345"`
`$called-number`	text	Called party phone number	`$called-number ~ "866*"`
`$caller-name`	text	Caller party name	`$caller-name = "David Amado"`
`$called-name`	text	Called party name	`$called-name != "David Amado"`
`$participant-number`	text	Participant phone number (either caller or called party)	`$participant-number = "12345"`
`$participant-name`	text	Participant phone name (either caller or called party)	`$participant-name = "David Amado"`
`$user-name`	text	Name of the user, to whom the call is assigned	`$user-name = "David Amado"`
`$user-extension`	text	Extension of the user, to whom the call is assigned	`$user-extension = "12345"`
`$user-id`	uuid	ID of the user, to whom the call is assigned	`$user-id = "9ef4b87c-5446-499a-b712-44d3509c0576"`
`$group-name`	text	Name of the group, to whom the call is assigned	`$group-name = "Sales Department"`
`$group-id`	uuid	ID of the group, to whom the call is assigned	`$group-id = "9ef4b87c-5446-499a-b712-44d3509c0576"`
`$tenant-name`	text	Name of the tenant, to whom the call is assigned	`$tenant-name = "West Coast"`
`$tenant-id`	uuid	ID of the tenant, to whom the call is assigned	`$tenant-id = "9ef4b87c-5446-499a-b712-44d3509c0576"`
`$call-id`	uuid	ID of the call	`$call-id = "9ef4b87c-5446-499a-b712-44d3509c0576"`
`$parent-call-id`	uuid	ID of the parent call	`$parent-call-id = "9ef4b87c-5446-499a-b712-44d3509c0576"`
`$setup-time`	datetime	Date and time of the call start time	`$setup-time >= "2023-01-01 00:00:00"`
`$duration`	timedelta	Duration of the call	`$duration < "1:00"`
`$diretion`	text	Direction of the call. One of "inbound", "outbound", "internal" and "uknown"	`$direction = inbound`
`$sentiment-score`	number	Sentiment score of the call	`$sentiment-score < 0`
`$sentiment-agent-score`	number	Sentiment score of agent side of the call	`$sentiment-agent-score < -50`
`$sentiment-customer-score`	number	Sentiment score of customer side of the call	`$sentiment-agent-score >= 0`

`$sentiment-label`	text	Sentiment score label of the call. One of "very-negative", "negative", "neutral", "positive", "very-positive"	`$sentiment-label = "very-negative"`
`$sentiment-agent-label`	text	Sentiment score label of the call	`$sentiment-agent-label = "negative"`

Attribute	Type	Description	Example
`$sentiment-customer-label`	text	Sentiment score label of the call	`$sentiment-customer-label = "positive"`
`$topic-name`	text	Name of the topic, assigned to the call	`$topic-name = "Payment language"`
`$topic-id`	uuid	ID of the topic, assigned to the call	`$topic-id = "9ef4b87c-5446-499a-b712-44d3509c0576"`

Comparison operators

Table 2. Supported comparison operators for **text** attribute types

Comparison operator	Description	Example
`=`	Equal to	`$caller-number = "12345"`
`==`	Equal to	`$caller-number == "12345"`
`:`	Equal to	`$caller-number: "12345"`
`!=`	Not equal to	`$caller-number != "12345"`
`<>`	Not equal to	`$caller-number <> "12345"`
`>`	Greater than (by alphabetical order)	`$topic-name > "1. Payment language"`
`>=`	Greater than or equal to (by alphabetical order)	`$topic-name >= "3. Shipping problem"`
`<`	Less than (by alphabetical order)	`$topic-name < "1. Payment language"`
`<=`	Less than or equal to (by alphabetical order)	`$topic-name <= "3. Shipping problem"`
`~`	Simple pattern (case sensitive)	`$user-name ~ "John*"`
`~*`	Simple pattern (case insensitive)	`$user-name ~* "john*"`
`~~`	Regex pattern (case sensitive)	`$phone-number ~~ "800[0-9]{6}"`
`~~*`	Regex pattern (case insensitive)	`$phone-name ~~* "(john|marry)"`

Table 3. Supported comparison operators for **number** attribute types

Comparison operator	Description	Example
`=`	Equal to	`$sentiment-score = 50`
`==`	Equal to	`$sentiment-score == 50`
`:`	Equal to	`$sentiment-score: 50`

`!=`	Not equal to	`$sentiment-score != 50`
`<>`	Not equal to	`$sentiment-score <> 50`
`>`	Greater than	`$sentiment-score > 0`
`>=`	Greater than or equal to	`$sentiment-score >= -50`
`<`	Less than	`$sentiment-score < -50`
`<=`	Less than or equal to	`$sentiment-score <= 0`

Table 4. Supported comparison operators for **datetime** attribute types

Comparison operator	Description	Example
`=`	Equal to	`$setup-time = "2023-01-01 00:00:00"`
`==`	Equal to	`$setup-time == "2023-01-01 00:00:00"`
`:`	Equal to	`$setup-time: "2023-01-01 00:00:00"`
`!=`	Not equal to	`$setup-time != "2023-01-01 00:00:00"`
`<>`	Not equal to	`$setup-time <> "2023-01-01 00:00:00"`
`>`	Greater than	`$setup-time > "2023-01-01 00:00:00"`
`>=`	Greater than or equal to	`$setup-time >= "2023-01-01 00:00:00"`
`<`	Less than	`$setup-time < "2023-01-01 00:00:00"`
`<=`	Less than or equal to	`$setup-time <= "2023-01-01 00:00:00"`

Table 5. Supported comparison operators for **timedelta** attribute types

Comparison operator	Description	Example
`=`	Equal to	`$duration = "5:00"`
`==`	Equal to	`$duration == "5:00"`
`:`	Equal to	`$duration: "5:00"`
`!=`	Not equal to	`$duration != "5:00"`
`<>`	Not equal to	`$duration <> "5:00"`
`>`	Greater than	`$duration > "1:00"`
`>=`	Greater than or equal to	`$duration >= "0:15"`
`<`	Less than	`$duration < "15:00"`
`<=`	Less than or equal to	`$duration <= "0:15"`

Table 6. Supported comparison operators for **uuid** attribute types

Comparison operator	Description	Example
`=`	Equal to	`$topic-id = "79705555-5c4d-46b4-987d-7257fe2ae23e"`
`==`	Equal to	`$topic-id == "79705555-5c4d-46b4-987d-7257fe2ae23e"`
`:`	Equal to	`$topic-id: "79705555-5c4d-46b4-987d-7257fe2ae23e"`
`!=`	Not equal to	`$topic-id != "79705555-5c4d-46b4-987d-7257fe2ae23e"`
`<>`	Not equal to	`$topic-id <> "79705555-5c4d-46b4-987d-7257fe2ae23e"`
`>`	Greater than (by alphabetical order)	`$topic-id > "79705555-5c4d-46b4-987d-7257fe2ae23e"`
`>=`	Greater than or equal to (by alphabetical order)	`$topic-id >= "79705555-5c4d-46b4-987d-7257fe2ae23e"`
`<`	Less than (by alphabetical order)	`$topic-id < "79705555-5c4d-46b4-987d-7257fe2ae23e"`
`<=`	Less than or equal to (by alphabetical order)	`$topic-id <= "79705555-5c4d-46b4-987d-7257fe2ae23e"`

Multi-value attributes

Some attributes may have multiple values, like attributes `$topic-name`, `$participant-number`, `$user-name`, `$group-name`, etc.

For example, when multiple topics are assigned to a call, then the expression `$topic-name = "Payment language"` will evaluate to TRUE when either of the assigned topics is "Payment language".

Another example is the `$participant-number` attribute. Every call has at least two participants, so the expression `$participant-number = 1234` will evaluate to TRUE if either of the caller or called party phone numbers is 1234.

Comparing to a sub-expression

A call attribute can be compared to a sub-expression like:

- `$caller-number = (1234 OR 5679)`
- `$sentiment-label = ("negative" OR "very-negative")`

Comparing to another attribute

A call attribute can be compared to another attribute like:

- `$caller-number = $called-number`

Combining an attribute match with a text match

MQL expression can include both text and attribute expressions, like:

- `"thank you for calling" AND $direction = inbound`
- `(cancel NEAR order) AND $sentiment-score < -30`

Operators

MQL syntax supports various operators that you can use to build more complex queries. The following table briefly describes the MQL operators.

Operator	Use	Example
`AND`	The result of expression `x AND y` is `TRUE` when both `x` and `y` evaluate to `TRUE`.	`website AND problem`
`OR`	The result of expression `x OR y` is `TRUE` when either `x` or `y` evaluate to `TRUE`	`purchase OR buy`
`&`	A synonym of operator `AND`	`website & problem`
`|`	A synonym of operator `OR`	`purchase | buy`
`NOT`	The result of expression `NOT z` is `TRUE` when `z` evaluates to `FALSE`	`NOT "replacement order"`
`NOTIN`	The result of expression `x NOT y` is `TRUE` when `x` evaluates to `TRUE`, but `x` doesn't overlap with `y`	`problem NOTIN "not a problem`
`NEAR`	The result of expression `x NEAR:5 y` is `TRUE` when both `x` and `y` evaluate to `TRUE` and a distance between them is no more than 5	`cancel NEAR:3 order`
`NOTNEAR`	The result of expression `x NOTNEAR:5 y` is `TRUE` when `x` evaluates to `TRUE` and `y` either evaluates to `FALSE` or is found in a transcript at a distance of more than 5	`number NOTNEAR:3 phone`
`ONEAR`	The result of expression `x ONEAR:5 y` is `TRUE` when both `x` and `y` evaluate to `TRUE` and a distance between them is no more than 5, and `x` appears in a transcript before `y`	`cancel ONEAR:3 order`
`AFTER`	The result of expression `x AFTER:5 y` is `TRUE` when both `x` and `y` evaluate to `TRUE` and `x` appears in a transcript before the `y` at a distance no more than 5	`R"[0-9]+" AFTER:20 "credit card"`
`POSBEFORE`	The result of expression `POSBEFORE:50 x` is `TRUE` when `x` evaluates to `TRUE` and `x` appears in a transcript in the first 50 words	`POSBEFORE:50 "my name is"`
`POSAFTER`	The result of expression `POSAFTER:50 x` is `TRUE` when `x` evaluates to `TRUE` and `x` appears in a transcript after the 50th word. A position can be negative, where `POSAFTER:-50` means *the last 50 words of a transcript*	`POSAFTER:-50 "Have a great day"`
`AGENT`	The result of expression `AGENT: x` is `TRUE` when `x` evaluates to `TRUE` and the matched phrase was spoken by agent	`AGENT: "my name is"`
`A`	A synonym to AGENT	`A: "my name is"`
`CUSTOMER`	The result of expression `CUSTOMER: x` is `TRUE` when `x` evaluates to `TRUE` and the matched phrase was spoken by customer	`CUSTOMER: "my name is"`
`C`	A synonym to CUSTOMER	`C: "my name is"`
`REPEATS`	The result of expression `REPEATS:5-10 x` is `TRUE` when `x` evaluates to `TRUE` and appears in a transcript between 5 to 10 times	`REPEATS:5-10 "great"`

Grouping

Multiple expressions can be grouped with parentheses to form a more complex expression.

Examples:

Expression	Description
`(quick OR brown) AND fox`	matches "**quick fox**", "**brown fox**", but not "**grey fox**"
`cancel* NEAR (order|account)`	matches "**cancel order**", "**order is cancelled**", "**I am cancelling my account**", "**I want to cancel this order**"
`problem NOTIN ("no problem" OR "not a problem")`	matches "**This is a problem**", but ignores "**no problem at all**", and "**not a problem**"

Precedence rules

When no parentheses are present, then the operators are evaluated in the following order:

- `NOTNEAR`
- `ONEAR`
- `NEAR`
- `NOTIN`
- `REPEATS`
- `AGENT`, `A`, `CUSTOMER`, `C`
- `POSAFTER`
- `POSBEFORE`
- Metadata comparison characters, like `$sentiment < -10`
- `NOT`
- `AND`
- `OR`

Expression	Equivalent form
`quick OR brown AND fox`	`quick OR (brown AND fox)`
`quick NEAR brown AND fox`	`(quick NEAR brown) AND fox`

Default operator

If no operator is included between terms, then a default `ONEAR:5` operator is used:

Expression	Equivalent form
`brown fox`	`brown ONEAR:5 fox`

`(quick OR brown) fox`	`(quick OR brown) ONEAR:5 fox`
`quick OR brown fox`	`quick OR (brown ONEAR:5 fox)`

Note

The `ONEAR` operator has a higher priority than `OR` and `AND` (see the **Precedence rules** section).

For example, the expression `quick AND brown fox` is interpreted by the search engine as `quick AND (brown ONEAR:5 fox)`

Boolean operators (AND, OR, NOT, & and |)

Expression	Description
`quick OR brown`	matches "**quick fox**" and "**brown fox**"
`quick AND fox`	matches "**quick fox**"
`NOT brown AND fox`	matches "**quick fox**" but not "**brown fox**"

Synonyms & and |

Symbols `&` and `|` are synonyms for `AND` and `OR` respectively.

Expression	Equivalent form
`quick | brown`	`quick OR brown`
`quick & fox`	`quick AND fox`
`(quick | brown | grey ) & fox`	`(quick OR brown OR grey) AND fox`

When using `|` and `&` symbols, a space charter between words is optional. The following are valid expressions:

- `(quick | brown | grey ) & fox`
- `(quick|brown|grey)&fox`

Case in operator names

A case in the operator's name is important. `AND` is treated as an operator, while `and` is treated literally as a word "**and**" in the text "**what a beautiful and amazing day**".

Order of the matched terms

For boolean operators, and order of the matched words is not taken into account. If an order is important, then use the `ONEAR` operator.

Expression	Description
`quick AND fox`	matches both "**quick fox**" and "**fox quick**"

Distance between the matched terms

For boolean operators, a distance between words is not taken into account, i.e. the expression `x AND y` will evaluate to `TRUE` when terms `x` and `y` are found anywhere in a transcript. Use Quoted Term or operators `NEAR`, `ONEAR` and `NOTNEAR` if a distance is important.

Expression	Description
`quick AND fox`	matches both "**quick fox**" and "**quick dog was chasing a fox**"
`"quick fox"`	matches "**dog was chasing a quick fox**" bot not "**quick dog was chasing a fox**"
`quick NEAR:3 fox`	matches "**quick fox**" and "**quick and cute fox**" but not "**quick dog was chasing a fox**", because of a distance between **quick** and **fox** words is more than 3 words.

Proximity operators (NEAR, ONEAR, NOTNEAR, NOTIN, AFTER)

Proximity operators allow you to locate one searched term within a certain distance of another.

NEAR[:x]

Finds the phrase where the terms joined by the operator are within the specified number of words of each other. Where `x` is the maximum distance between the searched terms.

Key features:

- A distance parameter is optional. If omitted, a default distance of 5 is used, i.e. **NEAR** is equivalent to **NEAR:5**
- An order of the found terms is not taken into account, i.e. `brown NEAR fox` will match both "**dog is chasing brown fox**" and "**fox is chasing brown dog**".
- When chaining multiple operators, then parentheses must be used if the distance is not the same.

  For example, expressions `brown NEAR quick NEAR fox` and `brown NEAR:2 quick NEAR:2 fox` are both valid, but `brown NEAR:2 quick NEAR:5 fox` is not a valid expression because a distance is 2 in one case and 5 in another. Parentheses must be added to make such expression valid: `(brown NEAR:2 quick) NEAR:5 fox`

Expression	Description
`cancel* NEAR order`	Matches "**cancel my order**", "**order is cancelled**", but not "**cancel my account and then place an order**", because of a distance between **cancel** and **order** in the last example is more than default 5 words.
`cancel* NEAR:1 order`	Matches "**cancel order**", but not "**cancel my order**" because of distance between words is more than requested (1).
`brown NEAR quick NEAR fox`	Matches "**brown and quick fox**", but not "**brown fox**"

ONEAR[:x]

Similar to the **NEAR** operator, but an order of the matched terms is taken into account. For example, `brown NEAR fox` will match "**brown fox**" but not "**fox brown**".

Expression	Description
`cancel* ONEAR order`	Matches "**cancel my order**" but not "**order is cancelled**", because of the order of terms is important.

Key features:

- A distance parameter is optional. If omitted, a default distance of 5 is used, i.e. **ONEAR** is equivalent to **ONEAR:5**
- When chaining multiple operators, then parentheses must be used when the distance is not the same.

  For example, expressions `brown ONEAR quick ONEAR fox` and `brown ONEAR:2 quick ONEAR:2 fox` are both valid, but `brown ONEAR:2 quick ONEAR:5 fox` is not a valid expression because a distance is 2 in one case and 5 in another.

  Parentheses must be added to make such expression valid: `(brown ONEAR:2 quick) ONEAR:5 fox`

NOTNEAR[:x]

Syntax:

```
<term-1> NOTNEAR[:x] <term-2>
```

Operator `NOTNEAR` finds the term on the left side of the operator (`<term-1>`) that is not near the term on the right side of the operator (`<term-2>`).

Expression	Description
`cancel* NOTNEAR account`	Matches "**cancel order**", "**order is cancelled**", but neither "**cancel my account**" not "**this account is cancelled**".
`cancel* NOTNEAR:1 account`	Matches "**cancel my bank account**" but not "**cancel account**", because of a required distance between terms is maximum 1.

Key features:

- A distance parameter is optional. If omitted, a default distance of 5 is used, i.e. **NOTNEAR** is equivalent to **NOTNEAR:5**
- An order of the found terms is not taken into account, i.e. `cancel* NOTNEAR account` will exclude both "**cancel account**" and "**account canceled**"
- Chaining of operator NOTNEAR is not supported.

Use parentheses to specifically group multiple expressions.

For example, `cancel* NOTNEAR bank* NOTNEAR account` must be rewritten as `cancel* NOTNEAR (bank* NOTNEAR account)`

NOTIN

Operator NOTIN allows matching terms that are not part of a longer term. For example, you would like to find the word "**problem**", but not when it is part of the phrase "**not a problem**".

Examples:

- `problem NOTIN "not a problem"`
- `problem NOTIN "no problem"`
- `problem NOTIN ((no|not) ONEAR problem)`
- `problem NOTIN no* ONEAR:2 problem`

AFTER[:x]

Finds the phrase that appears in a transcript after another phrase.

An optional argument after the colon symbol (`x`) is the maximum distance between the searched terms.

Key features:

- A distance parameter is optional. If omitted, a default distance of 5 is used, i.e. **AFTER** is equivalent to **AFTER:5**
- The operator `AFTER` is partially equivalent to the `ONEAR` operator, i.e. the search expression `you AFTER thank` can be replaced with `thank ONEAR you`.

But there is one key difference between the `AFTER` and `ONEAR` operators: the result of the ONEAR expression is a whole matched text as a match ("thank you" in our example), while the result of the AFTER expression is the left term only ("you" in our example).

This becomes very handy when using data redaction functionality with search expressions like `R"[0-9]+" AFTER "credit card"`, which has a purpose of redacting digits from audio recording and transcript while keeping the text "credit card" intact.

The equivalent expression `"credit card" ONEAR R"[0-9]+"` would redact both the phrase `"credit card", digits and any other text in between these two found terms.

Expression	Description
`R"[0-9]+" AFTER "credit card"`	Matches "**123456**" in phrase "**my credit card number is 123456**" but not in "**my phone number is 123456**"

Count occurrences (REPEATS)

Operator **REPEATS** finds the term, that occurs the requested number of times in a text. For example, it can be used to find the phrase where at least 8 digits are spoken.

Syntax:

```
REPEATS:N[-M] <term>
```

Where:

- `<term>` is a search expression, which can be a word, phrase or a complex expression like `(brown | quick)`
- `N` is the minimum number of occurrences of the term in the text
- `M` is the maximum number of occurrences of the term in the text. If omitted, then maximum M is equal to N, i.e `brown MATCHES: 2` is the same as `brown MATCHES: 2-2`

Examples:

- `REPEATS:5-10 (great|appreciate)`

In some cases, a Regex pattern matching can be used as an alternative to `REPEATS` operators. For example, to find consecutive digits in a text, use the expression like:

```
R"([0-9][ ]*){3,10}"
```

Such an expression will match digits 0 to 9 in a text, optionally, separated with a space character, and requires a minimum 3, maximum 10 digits.

It is partially equivalent to the following `REPEAT` query:

```
REPEATS:3-10 (0|1|2|3|4|5|6|7|8|9)
```

We say *partially* because the `REPEATS` operator matches whole words, while `REGEX` operator can match part within the words. For instance, both expression will successfully match the digits in the text "credit card number is 1 2 3 4 5 6 7 8".

But, if the digits appear in a text as a single word, like in "credit card number is 12345678", then the `REPEATS` operator would not match this text, while `REGEX` operator successfully matches it.