IBM Security QRadar
Version 7.2.6

Ariel Query Language Guide

Product information

This document applies to IBM QRadar Security Intelligence Platform V7.2.6 and subsequent releases unless superseded by an updated version of this document.

Contents

About this guide

The Ariel Query Language (AQL) Guide provides you with information for using the AQL advanced searching and API.

Intended audience

System administrators who view event or flow data that is stored in the Ariel database.

Technical documentation

To find IBM® Security QRadar® product documentation on the web, including all translated documentation, access the IBM Knowledge Center (http://www.ibm.com/support/knowledgecenter/SS42VS/welcome).

For information about how to access more technical documentation in the QRadar products library, see Accessing IBM Security Documentation Technical Note (www.ibm.com/support/docview.wss?rs=0&uid=swg21614644).

Contacting customer support

For information about contacting customer support, see the Support and Download Technical Note (http://www.ibm.com/support/docview.wss?uid=swg21616144).

Statement of good security practices

IT system security involves protecting systems and information through prevention, detection and response to improper access from within and outside your enterprise. Improper access can result in information being altered, destroyed, misappropriated or misused or can result in damage to or misuse of your systems, including for use in attacks on others. No IT system or product should be considered completely secure and no single product, service or security measure can be completely effective in preventing improper use or access. IBM systems, products and services are designed to be part of a lawful comprehensive security approach, which will necessarily involve additional operational procedures, and may require other systems, products or services to be most effective. IBM DOES NOT WARRANT THAT ANY SYSTEMS, PRODUCTS OR SERVICES ARE IMMUNE FROM, OR WILL MAKE YOUR ENTERPRISE IMMUNE FROM, THE MALICIOUS OR ILLEGAL CONDUCT OF ANY PARTY.

Please Note:

Use of this Program may implicate various laws or regulations, including those related to privacy, data protection, employment, and electronic communications and storage. IBM Security QRadar may be used only for lawful purposes and in a lawful manner. Customer agrees to use this Program pursuant to, and assumes all responsibility for complying with, applicable laws, regulations and policies. Licensee represents that it will obtain or has obtained any consents, permissions, or licenses required to enable its lawful use of IBM Security QRadar.

Ariel Query Language (AQL)

The Ariel Query Language (AQL) is a structured query language that you use to communicate with the Ariel databases. Use AQL to manage event and flow data from the Ariel database.

Ariel Query Language (AQL) deprecated versions

Ariel Query Language (AQL) V1 and V2 are deprecated.

The command-line script, **`/opt/qradar/bin/arielClient`** is deprecated. The following warning message is displayed both before and after the results are returned:

```
WARNING: AQL V1 and V2 will be deprecated in the future.
 For information about using AQL V3, see the product documentation.
```

During your migration to AQL V3, you can suppress the warning message by typing: `/opt/qradar/bin/arielClient | grep -v WARNING`

The Python client and the Advanced search option use AQL V3.

AQL fields changed in AQL V3

Ariel Query Language (AQL) V2 is deprecated in QRadar V7.2.4 and later. Some Ariel database fields were changed or removed in AQL V3. If you have queries that use these fields, you must replace them.

This table shows the new Ariel database fields.

Table 1. Fields that were replaced in AQL V3

Field name (AQL V2)	Replacement function name (AQL V3)
destinationAssetName	AssetHostname
deviceGroup	LogSourceGroupName
sourceAssetName	AssetHostname
eventDescription	QidName
destinationNetwork	NetworkName
endDate	DateFormat
endDateFormatted	DateFormat
eventProcessor	Processorname
identityUsername	AssetUser
identityMAC	AssetProperty
identityHostName	AssetHostname
identityNetBiosName	AssetHostname
identityGroupName	AssetProperty
identityExtendedField	AssetProperty
deviceDate	DateFormat
payloadHex	UTF8
protocol	ProtocolName
sourceNetwork	NetworkName
startDate	DateFormat
startDateFormatted	DateFormat
destinationAssetName	AssetHostname

Table 1. Fields that were replaced in AQL V3 (continued)

Field name (AQL V2)	**Replacement function name (AQL V3)**
`sourceAssetName`	`AssetHostname`
`destinationNetwork`	`NetworkName`
`sourceNetwork`	`NetworkName`
`application`	`ApplicationName`
`destinationPayloadHex`	`UTF8`
`firstPacketDate`	`DateFormat`
`eventProcessorId`	`ProcessorName`

This lists shows the Ariel database fields that were removed.

- partialorMatchList
- qidNumber
- token
- destinationHost
- destinationIPSearch
- destinationPortNA
- sourceHost
- sourceIPSearch
- sourcePortNA
- destinationDscpOnly
- anyDestinationFlag
- smallDestinationPayload
- smallDestinationPayloadHex
- destinationPrecedanceOnly
- lastPacketDate
- localHost
- remoteHost
- sourceDscpOnly
- anySourceFlag
- sourcePayloadHex
- smallSourcePayload
- smallSourcePayloadHex
- sourcePrecedanceOnly
- sourceHostString
- destinationHostString
- destinationNetwork
- application
- sourceNetwork
- smallPayload
- smallPayloadHex
- quickSearchMatches
- bitsPerSecond
- srcBitsPerSecond
- dstBitsPerSecond
- bytesPerSecond

- bytesPerPacket
- srcBytesPerPacket
- dstBytesPerPacket
- destinationByteRatio
- destinationPacketRatio
- packetsPerSecond
- sourceByteRatio
- sourcePacketRatio
- totalBytes
- totalPackets
- retentionBucket
- properLastPacketTime
- properLastPacketDate

AQL functions

Use Ariel Query Language (AQL) built-in functions to do calculations on data in the Ariel database.

Note: When you build an AQL query, if you copy text that contains single quotation marks from any document and paste the text into IBM Security QRadar, your query will not parse. As a workaround, you can paste the text into QRadar and retype the single quotation marks, or you can copy and paste the text from the IBM Knowledge Center.

Table 2. Basic functions

Operator	Description	Example
`LONG`	Converts a value that represents a number into a long integer.	`LONG('1234')`
`DOUBLE`	Converts a value that represents a number into a double.	`DOUBLE('1234')`
`STR`	Converts any parameter to a string.	`STR(sourceIP)`
`STRLEN`	Returns the length of this string.	`STRLEN(userName)`
`STRPOS`	Returns the position (index - starts at zero) of a string within another string. Can optionally specify an additional parameter to indicate at what position (index) to start looking for the specified pattern.	`STRPOS(username, 'test')`, `STRPOS(username, 'test', 5)`
`SUBSTRING`	Copies a range of characters into a new string.	`SUBSTRING(userName, 0, 3)`
`CONCAT`	Concatenates all passed strings into 1 string.	`CONCAT(userName, STR(sourceIP))`
`PARSEDATETIME`	Returns the current time, which is expressed as milliseconds since the time 00:00:00 Coordinated Universal Time (UTC) on January 1, 1970.	`PARSEDATETIME('1 week ago')`
`DATEFORMAT`	Formats a time, which is expressed as milliseconds since the time 00:00:00 Coordinated Universal Time (UTC) on January 1, 1970 to a user-readable form.	`DATEFORMAT(startTime, 'YYYY-MM-DD HH:mm:ss') as StartTime`
`NOW`	Returns the current time that is expressed as milliseconds since the time 00:00:00 Coordinated Universal Time (UTC) on January 1, 1970.	`NOW()`
`UTF8`	Returns the UTF8 string of a byte array.	`UTF8(payload)`
`UPPER`	Returns an all uppercase representation of a string.	`UPPER(username)`

Table 2. Basic functions (continued)

Operator	Description	Example
`LOWER`	Returns an all lowercase representation of a string.	`LOWER(username)`
`REPLACEFIRST`	Match a regex and replace the first match with text. Replaces the first subsequence(arg2) of the input sequence that matches the pattern(arg1) with the given replacement string(arg3).	`REPLACEFIRST('\d{16}', username, 'censored')`
`REPLACEALL`	Match a regex and replace all matches with text. Replaces every subsequence(arg1) of the input sequence that matches the pattern(arg1) with the given replacement string(arg3).	`REPLACEALL('\d{16}', username, 'censored')`

Table 3. Aggregate functions

Operator	Information	Example
`GROUP BY`	Creates an aggregate on one or more columns.	`SELECT sourceIP, COUNT(*) from events group by sourceIP, destinationIP`
`COUNT`	Returns the count of the rows in the aggregate.	`SELECT sourceIP, COUNT(*) from events group by sourceIP`
`UNIQUECOUNT`	Returns the unique count of the value in the aggregate.	`SELECT sourceIP, UNIQUECOUNT (category) from events group by sourceIP`
`FIRST`	Returns the first entry of the rows in the aggregate.	`SELECT sourceIP, FIRST(magnitude) from events group by sourceIP`
`LAST`	Returns the last entry of the rows in the aggregate	`SELECT sourceIP, LAST(magnitude) from events group by sourceIP`
`SUM`	Returns the sum of the rows in the aggregate.	`SELECT sourceIP, SUM(sourceBytes) from flows group by sourceIP`
`AVG`	Returns the average value of the rows in the aggregate.	`SELECT sourceIP, AVG(magnitude) from events group by sourceIP`
`MIN`	Returns the minimum value of the rows in the aggregate.	`SELECT sourceIP, MIN(magnitude) from events group by sourceIP`
`MAX`	Returns the maximum value of the rows in the aggregate.	`SELECT sourceIP, MAX(magnitude) from events group by sourceIP`
`STDEV`	Returns the Sample Standard Deviation value of the rows in the aggregate	`SELECT sourceIP, STDEV(magnitude) from events group by sourceIP`
`STDEVP`	Returns the Population Standard Deviation value of the rows in the aggregate	`SELECT sourceIP, STDEVP(magnitude) from events group by sourceIP`
`HAVING`	Allows operators on the result of a grouped by column.	`SELECT sourceIP, MAX(magnitude) as MAG from events group by sourceIP HAVING MAG > 5`

Table 4. External functions

Name	Description	Argument type	Description
`HostName`	Looks up a log source ID or a flow source ID.	`NUMERIC`	Log source ID or the flow source ID.

`AssetHostname`	Looks up a host name of an asset at a point in time.
	
	Domain can optionally be specified in order to target an asset on a particular domain.
		`VARCHAR`
		`DOUBLE`
		`BIGINT`
			IP address, Time stamp
			
			Optional: If not specified, uses `NOW()`
			
			Optional: Domain ID

Table 4. External functions (continued)

Name	Description	Argument type	Description

`AssetProperty`	Looks up a property for an asset.
	
	Domain can optionally be specified in order to target an asset on a particular domain.
		`VARCHAR`
		`OTHER`
		`BIGINT`
			IP address, Property name
			
			Optional: Domain ID

`AssetUser`	Looks up a user for an asset at a point in time.
	
	Domain can optionally be specified in order to target an asset on a particular domain.
		`VARCHAR`
		`DOUBLE`
		`BIGINT`
			IP address, Timestamp
			
			Optional: If not specified, uses `NOW()`
			
			Optional: Domain ID

`MatchesAsset Search`	If the asset is contained in the results of the asset saved search it returns true.	`VARCHAR` `VARCHAR`	IP address, Saved Search Name
`ReferenceMap`	Looks up the value for a key in a reference map.	`JAVA_OBJECT` `JAVA_OBJECT`	String, String **Example:** `ReferenceMap ('IPLookup', 'userName')`

`ReferenceTable`	Looks up the value for a column key in a table that is identified by a table key in a specific reference table collection.	`VARCHAR`
		`JAVA_OBJECT`
		`JAVA_OBJECT`
			String, String, String (or IP address)
			Example:
			`ReferenceTable ('testTable', 'numKey', '100.10.10.1') or ReferenceTable ('testTable', 'numKey', sourceIP)`

`Reference MapSet Contains`	If a value is contained in a reference set that is identified by a key in a specific reference map of set it returns `true`.	`VARCHAR`
		`JAVA_OBJECT`
		`JAVA_OBJECT`
			String, String, String
			Example:
			`ReferenceMap SetContains( 'RiskyUsersForIps', 'sourceIP', 'userName')`

`ReferenceSet Contains`	If a value is contained in a specific reference set, it returns `true`.	`VARCHAR` `JAVA_OBJECT`	String, String **Example:** `ReferenceSetContains ('MySet', 'SourceIP')`
`CategoryName`	Looks up the name of a category by its ID.	`NUMERIC`	Category ID
`LogSource Group Name`	Looks up the name of a log source group by its log source group ID.	`NUMERIC`	Device group list **Example:** `LogSourceGroupName(deviceGroupList)`
`QidDescription`	Looks up the description of a QID by its QID.	`NUMERIC`	QID
`QidName`	Looks up the name of a QID by its QID.	`NUMERIC`	QID
`Application Name`	Returns the name of a flow application.	`NUMERIC`	Application ID
`LogSource Name`	Looks up the name of a log source by its log source ID.	`NUMERIC`	Log source ID **Example:** `LogSourceName(logSourceId)`
`LogSource Type Name`	Looks up the name of a log source type by its device type.	`NUMERIC`	Device type **Example:** `LogSourceTypeName(deviceType)`

Table 4. External functions (continued)

Name	Description	Argument type	Description
`UTF-8`	Returns the UTF-8string.	`VARBINARY`	A byte array **Example:** Payload
`StrLen`	Returns the length of this string.	`VARCHAR`	String
`Str`	Converts parameter to string.	`JAVA_OBJECT`	String

`SubString`	Copies a range of characters into a new string.	`VARCHAR`
		`NUMERIC`
		`NUMERIC`
			A String, a start that is offset, and a length

`Concat`	Concatenates all passed strings into 1 string.	`VARCHAR`
		`NUMERIC`
		`NUMERIC`
			List of strings

`ParseDate time`	Returns the current time, which is expressed as milliseconds since the time 00:00:00 Coordinated Universal Time (UTC) on January 1, 2014.	`VARCHAR`	A String that represents a date and time
`Now`	Returns the current time, which is expressed as milliseconds since the time 00:00:00 Coordinated Universal Time (UTC) on January 1, 2014.	`NULL`	None
`ProtocolName`	Returns the name of a protocol, which is based on a protocol ID number.	`NUMERIC`	Protocol ID number

`InOffense`	If an event or flow belongs to the specified offense, it returns `true`.	`NUMERIC`	Offense ID
			Example:
			`SELECT * FROM events WHERE InOffense(123)`
			`SELECT * FROM flows WHERE InOffense(123)`

`InCIDR`	If the IP/column, specified is contained in, or equal to, the specified IP/CIDR, it returns `true`.	`VARCHAR, OTHER`	IP/CIDR, IP address **Example:** `...WHERE InCIDR('172.16.0.0/16', sourceip) AND ...`
`NetworkName`	Looks up the network name from the network hierarchy for the `Host` that is passed in.	`OTHER`	Host property **Example:** `NetworkName(sourceip)`
`RuleName`	Returns one or more rule names that are based on the rule ID or IDs that are passed in.	`INTEGER`	A single rule ID, or a list of rule IDs. **Example:** `RuleName(creEventList), RuleName(1033)`
`Long`	Parses a string that represents a number into a Long (integer) data type.	`VARCHAR`	A string that represents a number. **Example:** `Long('1234')`
`Double`	Parses a string that represents a number into a Double (integer) data type.	`VARCHAR`	A string that represents a number. **Example:** `Double('1234')`
`DomainName`	Looks up the domain name based on domain ID.	`NUMERIC`	domain ID **Example:** DomainName(domainID)

Logical and comparative operators

Logical operators are used in AQL statements to determine any equality or difference between values. By using logical operators in the WHERE clause of an AQL statement, the results returned are restricted/filtered to those that match the conditions in the WHERE clause. The following table lists the supported operators.

Note: When you build an AQL query, if you copy text that contains single quotation marks from any document and paste the text into IBM Security QRadar, your query will not parse. As a workaround, you can paste the text into QRadar and retype the single quotation marks, or you can copy and paste the text from the IBM Knowledge Center.

Table 5. Operators for the Ariel API

Operator	Information	Example
`=`	Compares 2 values and returns true if they are equal.	`...WHERE sourceIP = destinationIP`
`!=`	Compares 2 values and returns true if they are not equal.	`...WHERE sourceIP != desintationIP)`
`( and )`	Use brackets to nest components of a `WHERE` or `HAVING` clause to create complex Boolean expressions.	`...WHERE ( sourceIP = destinationIP ) AND ( sourcePort = destinationPort )`
`< and <=`	Compares two values and returns true if the left value is less than or, less than or equal to, the right value.	`...WHERE sourceBytes < 64 and destinationBytes <= 64`
`> and >=`	Compares two values and returns true if the left value is greater than or, greater than or equal to, the right value.	`...WHERE sourceBytes > 64 and destinationBytes >= 64`
`*`	Multiplies 2 values and returns the result.	`...WHERE sourceBytes * 1024 < 1`
`/`	Divides 2 values and returns the result.	`...WHERE sourceBytes / 8 > 64`
`+`	Adds 2 values and returns the result.	`...WHERE sourceBytes + destinationBytes < 64`
`-`	Subtracts 1 value from another and returns the result.	`...WHERE sourceBytes - destinationBytes > 0`
`^`	Takes a value and raises it to the specified power and returns the result.	`...WHERE sourceBytes ^ 2 < 256`
`%`	Takes the modulo of a value and returns the result.	`...WHERE sourceBytes % 8 == 7`
`AND`	Takes the left side of a statement and the right side of a statement and returns true if both are true.	`...WHERE ( sourceIP = destinationIP ) AND ( sourcePort = destinationPort )`
`OR`	Takes the left side of a statement and the right side of a statement and returns true if either one is true.	`...WHERE ( sourceIP = destinationIP ) OR ( sourcePort = destinationPort )`

Table 5. Operators for the Ariel API (continued)

Operator	Information	Example
`NOT`	Takes in a statement and returns true if the statement evaluates to false.	`...WHERE NOT ( sourceIP = destinationIP )`
`IS NULL`	Takes in a value and returns true if the value is null.	`...WHERE userName  IS NULL`
`NOT NULL`	Takes in a value and returns true if the value is not null.	`...WHERE userName  IS NOT NULL`
`BETWEEN (X,Y)`	Takes in a left side and two values and returns true if the left side is between the two values.	`...WHERE magnitude BETWEEN 1 AND 5`

`LIMIT`	Limits the number of results to the provided number.	Example 1
		`...WHERE magnitude > 5 LIMIT 10`
		
		Example 2
		`SELECT * FROM events LIMIT 100`
		`START '2015-10-28 10:00'`
		`STOP '2015-10-28 11:00'`
		
		Note: Place the LIMIT clause before a START and STOP clause.

`ORDER BY (ASC,DESC)`	Orders the result set by the provided columns.	`SELECT * FROM EVENTS ORDER BY sourceIP DESC`
`COLLATE`	Parameter to order by that allows a BCP47 language tag to collate.	`SELECT * FROM EVENTS ORDER BY sourceIP DESC COLLATE 'de-CH'`
`INTO`	Creates a named cursor that contains results that can be queried at a different time.	`SELECT * FROM EVENTS INTO 'MyCursor' WHERE....`

`START`	You can pass a time interval to `START` selecting data (from time), in the following formats:
	`yyyy-MM-dd HH:mm`
	`yyyy-MM-dd HH:mm:ss`
	`yyyy/MM/dd HH:mm:ss`
	`yyyy/MM/dd-HH:mm:ss`
	`yyyy:MM:dd-HH:mm:ss`
	
	The *timezone* is represented by 'z or Z' in the following formats:
	`yyyy-MM-dd HH:mm'Z'`
	`yyyy-MM-dd HH:mm'z'`
	
	Use in combination with `STOP`.
		Example 1
		`...WHERE userName IS NULL`
		`START '2014-04-25 15:51'`
		`STOP '2014-04-25 17:00'`
		
		The results returned from example 1 are from `'2014-04-25 15:51:00' to '2014-04-25 16:59:59'`
		
		Example 2
		`...WHERE userName IS NULL`
		`START '2014-04-25 15:51:20'`
		`STOP '2014-04-25 17:00:20'`
		
		The results returned from example 2 are from `'2014-04-25 15:51:00' to '2014-04-25 17:00:59'`
		
		Any format can be used with the `PARSEDATETIME` function, for example,
		`Select * from events`
		`START PARSEDATETIME('1 hour ago')`
		`STOP PARSEDATETIME('now')`
		
		`STOP` is optional. If you don't include it in the query, the STOP time is = `now()`.

Table 5. Operators for the Ariel API (continued)

Operator	Information	Example

`STOP`	You can pass a time interval to `STOP` selecting data (end time), in the following formats:
	`yyyy-MM-dd HH:mm`
	`yyyy-MM-dd HH:mm:ss`
	`yyyy/MM/dd HH:mm:ss`
	`yyyy/MM/dd-HH:mm:ss`
	`yyyy:MM:dd-HH:mm:ss`
	The *timezone* is represented by '`z` or `Z`' in the following formats:
	`yyyy-MM-dd HH:mm'Z'`
	`yyyy-MM-dd HH:mm'z'`
	Use in combination with `START`.
		`...WHERE userName IS NULL`
		`START  '2014-04-25 14:00'`
		`STOP '2014-04-25 16:00'`
		`...WHERE userName IS NULL`
		`START '2014-04-25 15:00:30'`
		`STOP '2014-04-25 15:02:30'`
		Any format can be used with the `PARSEDATETIME` function, for example,
		`Select * from events`
		`START PARSEDATETIME('1 day ago')`
		Even though `STOP` is not included in this query, the STOP time is = `now()`.

`LAST`	You can pass a time interval to select data from. Valid intervals are MINUTES, HOURS, and DAYS	`...WHERE userName IS NULL`
		` LAST 6 HOURS`

`LIKE`	Matches if the string passed, is `LIKE` the passed value. % is a wildcard.	`...WHERE userName LIKE '%bob%'`
`ILIKE`	Matches if the string passed, is LIKE the passed value in a case-insensitive manner. `%` is a wildcard.	`...WHERE userName ILIKE '%bob%'`

`MATCHES`	Matches if the string matches the provided regular expression.	`...WHERE userName MATCHES`
		` '^.bob.$'`

`IMATCHES`	Matches if the string matches the provided regular expression in a case-insensitive manner.	`...WHERE userName`
		` IMATCHES '^.bob.$'`

Table 5. Operators for the Ariel API (continued)

Operator	Information	Example

`TEXT SEARCH`	Full-text search for the passed value.
	
	TEXT SEARCH is valid with AND operators. You can't use TEXT SEARCH with OR or other operators; otherwise you will get a syntax error.
	
	Place TEXT SEARCH in the first position of the WHERE clause. Placing TEXT SEARCH in any other position of the WHERE clause can result in errors. See the WHERE TEXT SEARCH example that shows the correct order.
	
	You can also do full-text searches by using the Quick filter in the QRadar user interface. For information about Quick filter functions, see the *IBM Security QRadar SIEM Users Guide*
		`...WHERE TEXT SEARCH 'firewall' AND ...`
		
		`SELECT sourceip,url from events WHERE TEXT SEARCH 'download.cdn.mozilla.net' AND sourceip='192.168.1.1' START '2015-01-30 16:10:12' STOP '2015-02-22 17:10:22'`

Examples of logical and comparative operators

- To sort events that are unparsed, type the following query:

```
SELECT * FROM events WHERE payload = 'false'
```

- To sort events to find a specific source IP address that has an offense, type the following query:

```
SELECT * FROM events WHERE sourceIP = '231.12.37.17' AND
hasOffense = 'true'
```

- You can do a **Quick filter** search in AQL. To sort events for "firewall", type the following query:

```
SELECT QIDNAME(qid) AS EventName, * from events where TEXT SEARCH 'firewall'
```

Event, flow and simarc fields for AQL queries

Use the Ariel Query Language (AQL) to retrieve specific fields from the events, flows and simarc table in the Ariel database.

Supported flow fields for AQL queries

The flow fields that you can query are listed in the following table.

Table 6. Supported flow fields for AQL queries

Field name	Description
applicationId	Application ID
category	Category
credibility	Credibility
destinationASN	Destination ASN
destinationBytes	Destination bytes
destinationDSCP	Destination DSCP

Table 6. Supported flow fields for AQL queries (continued)

Field name	Description
destinationFlags	Destination flags
destinationIP	Destination IP
destinationIfIndex	Destination if index
destinationPackets	Destination packets
destinationPayload	Destination payload
destinationPort	Destination port
destinationPrecedence	Destination precedence
destinationTOS	Destination QoS
destinationv6	IPv6 destination
processorID	Event processor ID
fullMatchList	Full match list
firstPacketTime	First packet time
flowBias	Flow bias

flowDirection	Flow direction
	`local-to-local (L2L)`
	`local-to-remote (L2R)`
	`remote-to-local (R2L)`
	`remote-to-remote (R2R)`

flowInterfaceID	Flow interface ID
flowSource	Flow Source
flowType	Flow type
geographic	Matches geographic location
hasDestinationPayload	Has destination payload
hasOffense	Has offense payload
hasSourcePayload	Has source payload
icmpCode	Icmp code
icmpType	ICMP type or code
flowInterface	Flow interface
intervalId	Interval ID
isDuplicate	Duplicate event
lastPacketTime	Last packet time
partialMatchList	Partial match list
protocol	Protocol
protocolId	Protocol ID
qid	Qid
relevance	Relevance
retentionBucket	Retention bucket dummy
severity	Severity
sourceASN	Source ASN

Table 6. Supported flow fields for AQL queries (continued)

Field name	**Description**
sourceBytes	Source bytes
sourceDSCP	Source DSCP
sourceFlags	Source flags
sourceIP	Source IP
sourceIfIndex	Source if index
sourcePackets	Source packets
sourcePayload	Source payload
sourcePort	Source port
sourcePrecedence	Source precedence
sourcev6	IPv6 source
startTime	Start time
viewObjectPair	View object pair

Supported event fields for AQL queries

The event fields that you can query are listed in the following table.

Table 7. Supported event fields for AQL queries

Field name	**Description**
category	Low-level category
creEventList	Matched custom rule
credibility	Credibility
destinationMAC	Destination MAC
destinationPort	Destination port
destinationv6	IPv6 destination
deviceTime	Log source time
deviceType	Log source type
domainID	Domain ID **Note:** QRadar Log Manager only
duration	Duration
endTime	End time
eventCount	Event count

eventDirection	Event direction:
	
	`local-to-Local (L2L)`
	
	`local-to-remote (L2R)`
	
	`remote-to-local (R2L)`
	
	`remote-to-remote (R2R)`

processorId	Event Processor ID
hasIdentity	Has identity
hasOffense	Associated with offense

Table 7. Supported event fields for AQL queries (continued)

Field name	Description
highLevelCategory	High-level category
isCREEvent	Is custom rule event
magnitude	Magnitude
payload	Payload
postNatDestinationIP	Destination IP after NAT
postNatDestinationPort	Destination port after NAT
postNatSourceIP	Source IP after NAT
postNatSourcePort	Source port after NAT
preNatDestinationIP	Destination IP before NAT
preNatDestinationPort	Destination port before NAT
preNatSourceIP	Source IP before NAT
preNatSourcePort	Source port before NAT
protocolID	Protocol
qid	Event name ID
relevance	Relevance
severity	Severity
sourceIP	Source IP
sourceMAC	Source MAC
sourcePort	Source port
sourcev6	IPv6 source
startTime	Start time
isunparsed	Event is unparsed
userName	User name

Supported simarc fields for AQL queries

The simarc fields that you can query are listed in the following table.

Table 8. Supported simarc fields for AQL queries

Field name	Description
destinationPort	Destination port key creator
destinationType	Destination type key creator
deviceId	Device key creator
direction	Direction key creator
eventCount	Event count key creator
eventFlag	Flag key creator
applicationId	Application ID key creator
flowCount	Flow count key creator
destinationBytes	Destination bytes key creator
flowSource	Flow source key creator
sourceBytes	Source bytes key creator

Table 8. Supported simarc fields for AQL queries (continued)

Field name	Description
lastPacketTime	Time key creator
protocolId	Protocol key creator
source	Source key creator
sourceType	Source type key creator
sourceRemoteNetwork	Source remote network key creator
destinationRemoteNetwork	Destination remote network key creator
sourceCountry	Source geographic key creator
destinationCountry	Destination geographic key creator
destination	Destination key creator
creEventList	Normalized event properties CRE event list
partialMatchList	Normalized event properties partial match list

SELECT statement

Use the SELECT statement to retrieve specific data from the events or flows table in the Ariel database. A SELECT operation is called a *query*.

Syntax

Note: When you build an AQL query, if you copy text that contains single quotation marks from any document and paste the text into IBM Security QRadar, your query will not parse. As a workaround, you can paste the text into QRadar and retype the single quotation marks, or you can copy and paste the text from the IBM Knowledge Center.

```
SELECT selectList
     FROM joinClauses
     [WHERE searchCondition]
     [GROUP BY groupClause]
     [ORDER BY orderClause]
```

Usage

A SELECT statement can include one or more fields from the flow or event tables. Use an asterisk, *, to denote all columns. All field names are case-sensitive. However, SELECT and FROM statements are not case-sensitive.

Note: Place the LIMIT clause before the START and STOP clauses, for example,

```
SELECT *
    FROM events
     LIMIT 100
     START '2015-10-28 10:00' STOP '2015-10-28 11:00'
```

Overriding the time settings passed to the AQL query

The SELECT statement supports an `arieltime` option, which overrides the time settings.

You can limit the time period for which an AQL query is evaluated.

You can use the START and STOP keywords.

You can also use the LAST keyword.

Example:

```
SELECT * FROM events LAST 15 MINUTES
SELECT * FROM events LAST 1 HOUR
SELECT * FROM events LAST 2 DAYS
```

Examples of SELECT statements that use CIDR ranges

You can also use SELECT statements for CIDR-based queries. To query by source IP address, `sourceIP`, or by destination IP address, destinationIP, use the following format:

```
SELECT <query item> FROM <flows|events> WHERE
<sourceCIDR|destinationCIDR> = '<CIDR Range>'
```

Example:

```
SELECT * FROM flows WHERE sourceCIDR = '10.100.100/24'
```

To return all flows that are coming from the 10.100.100 subnet or capture flows that are coming from and into the subnet, use the regular OR expression.

Example:

```
SELECT * FROM flows WHERE sourceCIDR = '10.100.100/24' OR
destinationCIDR = '10.100.100/24'
```

To query when `source IP` is contained in the 192.168.222.0/24 range, use the following format:

```
SELECT <query item> FROM <events> WHERE
<INCIDR> = '<INCIDR Range>'
```

Example:

```
SELECT * FROM events WHERE INCIDR('192.168.222.0/24', sourceIP)
```

To query when `source IP` is not contained in the 192.168.222.0/24 range, use the following format:

```
SELECT <query item> FROM <events> WHERE
<INCIDR> != '<INCIDR Range>'
```

Example:

```
SELECT * FROM events WHERE NOT INCIDR('192.168.222.0/24', sourceIP)
```

WHERE clause

Restrict your AQL queries by using WHERE clauses. The WHERE clause describes the filter criteria to apply to the query and filters the resulting view to accept only those events or flows that meet the specified condition.

Syntax

```
WHERE searchCondition
```

A *searchCondition* is a combination of logical and comparison operators that together make a test. Only those input rows that pass the test are included in the result.

Examples of WHERE clauses

The following query example shows events that have a severity level of greater than 9 are selected from a category.

```
SELECT sourceIP, category, credibility FROM events WHERE
severity > 9 AND category = 5013
```

You can change the order of evaluation by using parentheses. The search conditions that are enclosed in parentheses are evaluated first.

```
SELECT sourceIP, category, credibility FROM events WHERE
(severity > 9 AND category = 5013) OR (severity < 5 and
credibility > 8)
```

GROUP BY clause

Use the GROUP BY clause to aggregate your data. To provide meaningful results of the aggregation, usually, data aggregation is combined with arithmetic functions on remaining columns.

Syntax

```
GROUP BY groupClause
```

You can use aggregate functions in Ariel Query Language (AQL) queries to summarize information from multiple rows. The aggregate functions that are supported are shown in the following table.

Table 9. Aggregate functions

Function	Description
GROUP BY	Creates an aggregate on one or more columns.
COUNT	Returns the count of the rows in the aggregate.
UNIQUECOUNT	Returns the unique count of the value in the aggregate.
FIRST	Returns the first entry of the rows in the aggregate.
SUM	When used with numeric data, returns the sum of the values. When used with categorical data, it returns the union of the categorical values.
AVG	Returns the average value of the rows in the aggregate.
MIN(expr)	Returns the lowest value of the rows in the aggregate..
MAX(expr)	Returns the highest value of the rows in the aggregate.
HAVING	Allows operators on the result of a grouped by column.

Examples of GROUP BY clauses

The following query example shows IP addresses that sent more than 1 million bytes within all flows in a specific time.

```
select sourceIP, SUM(sourceBytes) from flows where sourceBytes >
1000000 group by sourceIP
```

The results might look similar to the following output.

```
-----------------------------------
| sourceIP | SUM_sourceBytes |
-----------------------------------
| 64.124.201.151 | 4282590.0 |
| 10.105.2.10 | 4902509.0 |
| 10.103.70.243 | 2802715.0 |
| 10.103.77.143 | 3313370.0 |
| 10.105.32.29 | 2467183.0 |
| 10.105.96.148 | 8325356.0 |
| 10.103.73.206 | 1629768.0 |
-----------------------------------
```

However, if you compare this information to a non-aggregated query, the output displays all the IP addresses that are unique, as shown in the following output:

```
------------------------------
| sourceIP | sourceBytes |
------------------------------
| 64.124.201.151 | 1448629 |
| 10.105.2.10 | 2412426 |
| 10.103.70.243 | 1793095 |
| 10.103.77.143 | 1449148 |
| 10.105.32.29 | 1097523 |
| 10.105.96.148 | 4096834 |
| 64.124.201.151 | 2833961 |
| 10.105.2.10 | 2490083 |
| 10.103.73.206 | 1629768 |
| 10.103.70.243 | 1009620 |
| 10.105.32.29 | 1369660 |
| 10.103.77.143 | 1864222 |
| 10.105.96.148 | 4228522 |
------------------------------
```

To view the maximum number of events, use the following syntax:

```
SELECT MAX(eventCount) FROM events
```

To view the number of average events from a source IP, use the following syntax:

```
SELECT AVG(eventCount) FROM events GROUP BY sourceIP
```

The output displays the following results:

```
---------------------------------
| sourceIP | protocol |
---------------------------------
| 64.124.201.151 | TCP.tcp.ip |
| 10.105.2.10 | UDP.udp.ip |
| 10.103.70.243 | UDP.udp.ip |
| 10.103.77.143 | UDP.udp.ip |
| 10.105.32.29 | TCP.tcp.ip |
| 10.105.96.148 | TCP.tcp.ip |
| 64.124.201.151 | TCP.tcp.ip |
| 10.105.2.10 | ICMP.icmp.ip |
---------------------------------
```

ORDER BY clause

Use the ORDER BY clause to sort the resulting view that is based on expression results. The order is sorted by ascending or descending sequence.

Syntax

```
ORDER BY orderClause
```

Only one field can be used in the ORDER BY clause. You can switch sorting between ascending or descending by appending the ASC or DESC keyword to the order by clause.

Combining GROUP BY and ORDER BY clauses to create data

To determine the top abnormal events or the most bandwidth-intensive IP addresses, you can combine GROUP BY and ORDER BY clauses in a single query. When you combine the clauses, you create data, such as TopN lists. For example, the following query displays the most traffic intensive IP address in descending order:

```
SELECT sourceIP, SUM(sourceBytes) FROM flows GROUP sourceIP
ORDER BY SUM(sourceBytes) DESC
```

Examples of ORDER BY clauses

To query AQL to return results in descending order. use the following syntax:

```
SELECT sourceBytes, sourceIP FROM flows WHERE sourceBytes >
1000000 ORDER BY sourceBytes
```

To display results in ascending order, use the following syntax:

```
SELECT sourceBytes, sourceIP FROM flows WHERE sourceBytes >
1000000 ORDER BY sourceBytes ASC
```

LIKE clause

Use the LIKE clause to retrieve partial string matches in the Ariel database.

Syntax

```
ORDER BY orderClause
```

You can search fields by using the LIKE clause.

The following wildcard options are supported by the Ariel Query Language (AQL):

Table 10. Supported wildcard options for LIKE clauses

Wildcard character	Description
%	Matches a string of zero or more characters
_	Matches any single character

Examples of LIKE clauses

To match names such as Joe, Joanne, Joseph, or any other name that begins with Jo, type the following query:

```
SELECT * FROM events WHERE userName LIKE 'jo%'
```

To match names that begin with Jo that are 3 characters long, such as, Joe or Jon, type the following query:

```
SELECT * FROM events WHERE userName LIKE 'Jo_'
```

You can enter the wildcard option at any point in the command, as shown in the following examples.

```
SELECT * FROM flows WHERE sourcePayload LIKE '%xyz'
SELECT * FROM events WHERE payload LIKE '%xyz%'
SELECT * FROM events WHERE payload LIKE '_yz'
```

Examples of string matching keywords

The keywords, ILIKE and IMATCHES are case-insensitive versions of LIKE and MATCHES.

```
SELECT qidname(qid) as test FROM events WHERE test LIKE 'Information%'
SELECT qidname(qid) as test FROM events WHERE test ILIKE 'inForMatiOn%'

SELECT qidname(qid) as test FROM events WHERE test MATCHES '.*Information.*'
SELECT qidname(qid) as test FROM events WHERE test IMATCHES '.*Information.*'
```

COUNT function

The COUNT function returns the number of rows that satisfy the WHERE clause of a SELECT statement.

If the SELECT statement does not have a WHERE clause, the COUNT function returns the total number of rows in the table.

Syntax

```
COUNT
```

Examples

To count all events with credibility equal to or greater than 9, type the following query:

```
SELECT COUNT() FROM events WHERE credibility >= 9
```

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Trademarks

indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at Copyright and trademark information (www.ibm.com/legal/copytrade.shtml).

The following terms are trademarks or registered trademarks of other companies:

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user’s session id for purposes of session management and authentication. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, See IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Index

Printed in USA