Note
Before you use this information and the product that it supports, read the information in “Notices” on page 69.

Product information
This document applies to IBM QRadar Security Intelligence Platform V7.3.1 and subsequent releases unless superseded by an updated version of this document.

© Copyright IBM Corporation 2013, 2017.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
Contents

About this guide ................................................................. v

1 What's new for users in AQL .............................................. 1

2 Ariel Query Language ...................................................... 5
   Ariel Query Language deprecated versions ................................ 5
   AQL fields changed in AQL V3 ........................................... 5
   Overview of Ariel Query Language ...................................... 7
   Best practices for using quotation marks in AQL queries ................ 9
   SELECT statement ......................................................... 10
   WHERE clause ............................................................ 11
   GROUP BY clause ......................................................... 12
   HAVING clause ........................................................... 14
   ORDER BY clause ........................................................ 14
   LIKE clause .................................................................. 15
   COUNT function ................................................................ 16
   AQL logical and comparison operators .................................... 17
   AQL data calculation and formatting functions ....................... 19
   AQL data aggregation functions ........................................ 24
   AQL data retrieval functions ............................................. 27
   Time criteria in AQL queries ............................................. 40
   AQL date and time formats .............................................. 43
   AQL subquery .................................................................. 44
   Grouping related events into sessions .................................... 45
      Transactional query refinements ...................................... 47
   Conditional logic in AQL queries ........................................ 52
   Bitwise operators in AQL queries ....................................... 53
   CIDR IP addresses in AQL queries ..................................... 56
   Custom properties in AQL queries ..................................... 56
   System performance query examples ................................... 57
   Events and flows query examples ....................................... 58
   Reference data query examples ....................................... 59
   User and network monitoring query examples ....................... 61
   Event, flow, and simarc fields for AQL queries ....................... 63

Notices .................................................................................. 69
   Trademarks ....................................................................... 70
   Terms and conditions for product documentation ..................... 70
   IBM Online Privacy Statement ......................................... 71
   General Data Protection Regulation .................................... 72

Index ...................................................................................... 73
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.
1 What's new for users in AQL

IBM Security QRadar V7.3.1 introduces new Ariel Query Language (AQL) functions and enhancements.

PARAMETERS REMOTESERVERS now includes the option to select servers in your search by specifying the ID or name of Event Processors

By using the ARIELSERVERS4EPNAME function with PARAMETERS REMOTESERVERS, you can specify an Event Processor by name in an AQL query; for example, PARAMETERS REMOTESERVERS=ARIELSERVERS4EPNAME('eventprocessor0', 'eventprocessor104')

By using the ARIELSERVERS4EPID function with PARAMETERS REMOTESERVERS; you can specify an Event Processor by ID in an AQL query, for example, PARAMETERS REMOTESERVERS=ARIELSERVERS4EPID(102)

By specifying an Event Processor, or servers that are connected to that Event Processor, you can run AQL queries faster and more efficiently.

When you have multiple servers in your organization and you know where the data that you’re looking for is saved, you can fine-tune the search to just the servers, clusters, or specific servers on Event Processors.

In the following example, you search only the servers that are connected to 'eventprocessor104'.

```
SELECT processorid,PROCESSORNAME(processorid), LOGSOURCENAME(logsourceid)
FROM events
GROUP BY logsourceid
PARAMETERS REMOTESERVERS=ARIELSERVERS4EPNAME ('eventprocessor104')
```

You can significantly reduce the load on your servers, run the query regularly, and get your results faster when you filter your query to search fewer servers.

Learn more about PARAMETERS REMOTESERVERS...

PARAMETERS EXCLUDESERVERS excludes servers from your AQL search

Avoid having to search all AQL servers by using PARAMETERS EXCLUDESERVERS to exclude specific servers:

- IP address; for example, PARAMETERS EXCLUDESERVERS='177.22.123.246:32006,172.11.22.31:32006'
- Event Processor name; for example, PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPNAME ('<eventprocessor_name>')
- Event Processor ID; for example, PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPID(<processor_ID>)

Searching only the servers that have the data that you require speeds up searches and uses less server resources.

Refine your query to exclude the servers that don't have the data that you're searching for. In the following example, you exclude servers that are connected to 'eventprocessorABC':

```
SELECT processorid,PROCESSORNAME(processorid), LOGSOURCENAME(logsourceid)
FROM events
GROUP BY logsourceid
PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPNAME ('eventprocessorABC')
```
If you refine multiple queries by using PARAMETERS EXCLUDESERVERS, you can reduce the load on your servers and get your results faster.

Specify the Event Processor name in an AQL query by using the ARIELSERVERS4EPNAME function with PARAMETERS REMOTESERVERS or PARAMETERS EXCLUDESERVERS

In an AQL query, you can include or exclude the servers that are connected to an Event Processor by using the ARIELSERVERS4EPNAME function to name an Event Processor in the query. For example, use the ARIELSERVERS4EPNAME function with PARAMETERS REMOTESERVERS to include eventprocessor_ABC in the query.

```sql
PARAMETERS REMOTESERVERS=ARIELSERVERS4EPNAME('eventprocessor_ABC')
```

For example, you might want to search to exclude all servers on a named Event Processor by using the ARIELSERVERS4EPNAME function with PARAMETERS EXCLUDESERVERS. In the following example eventprocessor_XYZ is excluded in the query.

```sql
PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPNAME('eventprocessor_XYZ')
```

Specify the Event Processor ID in an AQL query by using the ARIELSERVERS4EPID function with PARAMETERS REMOTESERVERS or PARAMETERS EXCLUDESERVERS

In an AQL query, you can include or exclude servers connected to an Event Processor by using the ARIELSERVERS4EPID function to specify the ID of an Event Processor in the query.

For example, include servers on the Event Processor that has the ID 101, PARAMETERS REMOTESERVERS=ARIELSERVERS4EPID(101)

For example, exclude servers on the Event Processor that has the ID 102, PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPID(102)

Filter your search by using the ARIELSERVERS4EPID function with the PARAMETERS REMOTESERVERS or PARAMETERS EXCLUDESERVERS to specify Event Processors by ID and their Ariel servers.

You can use the ARIELSERVERS4EPID function with PARAMETERS REMOTESERVERS and PARAMETERS EXCLUDESERVERS to specify Ariel servers that you want to include or exclude from your search.

You can also use the following query to list Ariel servers by Event Processor ID.

```sql
SELECT processorid, ARIELSERVERS4EPNAME(PROCESSORNAMe(processorid)) from events
```

Returns Ariel servers that are associated with an Event Processor that is identified by ID.

Here's an example of the output for the query, which shows the ID of the processor and the servers for that processor:
Learn more about the ARIELSERVERS4EPID function...

In an AQL query, you can specify Ariel servers that are connected to a named Event Processor by using the ARIELSERVERS4EPNAME function.

Use the ARIELSERVERS4EPNAME function with PARAMETERS REMOTESERVERS or PARAMETERS EXCLUDESERVERS to specify Ariel servers that you want to include or exclude from your search.

You can also use the following query to list Ariel servers by Event Processor name.

```
SELECT PROCESSORNAME(processorid), ARIELSERVERS4EPNAME(PROCESSORNAME(processorid)) from events
```

Here's an example of the output for the query, which shows the name of the processor and the servers:

```
eventprocessorABC  localhost:32011,172.16.158.95:32006
```

Learn more about the ARIELSERVERS4EPNAME function...

Use the COMPONENTID function to retrieve the ID for any named QRadar component and return data for that component.

For example, you can retrieve events for a named Event Processor. In the following example you retrieve events from eventprocessor0:

```
SELECT * from events where processorid = COMPONENTID('eventprocessor0')
```

PARSETIMESTAMP function parses the text representation of date and time and converts it to UNIX epoch time

Do time-based calculations easily in AQL when you convert time in text format to epoch time.

Include time-based calculations in your AQL queries and use the time-based criteria that you specify to return events that helps to enhance the security of your organization by making it easier to monitor user activity. For example, you might want to find out that the difference between user logout and re-login times is less than 30 minutes. If this timing seems suspicious, you can investigate further.

Learn more about the PARSETIMESTAMP function...

Retrieve information about the location and distance of IP addresses

Use geographical data that is provided by MaxMind to find information about the location and distance between IP addresses in QRadar.

The GEO::LOOKUP AQL function returns location data for a selected IP address.

The GEO::DISTANCE AQL function returns the distance, in kilometers, of two IP addresses.

Easily recognize the geographical origin of your data by organizing your data by location such as city or country instead of by IP address, and use the distance between IP addresses to evaluate the relative distance between your QRadar locations.

Learn more about using the GEO::LOOKUP and the GEO::DISTANCE functions...
Enhanced support for the AQL subquery

In QRadar V.7.2.8 and V.7.3.0, the subquery was accessible only by using API.

The subquery is now available for use in searches from the Log Activity or Network Activity tabs.

Learn more about the AQL subquery...

Enhanced support for the SESSION BY clause

In QRadar V.7.3.0 the SESSION BY clause was accessible only by using API.

The SESSION BY clause is now available for use in searches in QRadar.

Learn more about the AQL SESSION BY clause...
2 Ariel Query Language

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

Ariel Query Language 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.

Table 1 shows the new Ariel database fields.

<table>
<thead>
<tr>
<th>Field name (AQL V2)</th>
<th>Replacement function name (AQL V3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>destinationAssetName</td>
<td>AssetHostname</td>
</tr>
<tr>
<td>deviceGroup</td>
<td>LogSourceGroupName</td>
</tr>
<tr>
<td>sourceAssetName</td>
<td>AssetHostname</td>
</tr>
<tr>
<td>eventDescription</td>
<td>QidName</td>
</tr>
<tr>
<td>destinationNetwork</td>
<td>NetworkName</td>
</tr>
<tr>
<td>endDate</td>
<td>DateFormat</td>
</tr>
<tr>
<td>endDateFormatted</td>
<td>DateFormat</td>
</tr>
<tr>
<td>eventProcessor</td>
<td>ProcessName</td>
</tr>
<tr>
<td>identityUsername</td>
<td>AssetUser</td>
</tr>
<tr>
<td>identityHostName</td>
<td>AssetHostname</td>
</tr>
<tr>
<td>identityNetBiosName</td>
<td>AssetHostname</td>
</tr>
<tr>
<td>identityGroupName</td>
<td>AssetProperty</td>
</tr>
<tr>
<td>identityExtendedField</td>
<td>AssetProperty</td>
</tr>
<tr>
<td>deviceDate</td>
<td>DateFormat</td>
</tr>
<tr>
<td>payloadHex</td>
<td>UTF8</td>
</tr>
<tr>
<td>protocol</td>
<td>ProtocolName</td>
</tr>
<tr>
<td>sourceNetwork</td>
<td>NetworkName</td>
</tr>
<tr>
<td>startDate</td>
<td>DateFormat</td>
</tr>
<tr>
<td>startDateFormatted</td>
<td>DateFormat</td>
</tr>
<tr>
<td>destinationAssetName</td>
<td>AssetHostname</td>
</tr>
<tr>
<td>sourceAssetName</td>
<td>AssetHostname</td>
</tr>
<tr>
<td>destinationNetwork</td>
<td>NetworkName</td>
</tr>
</tbody>
</table>
Table 1. Fields that were replaced in AQL V3 (continued)

<table>
<thead>
<tr>
<th>Field name (AQL V2)</th>
<th>Replacement function name (AQL V3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceNetwork</td>
<td>NetworkName</td>
</tr>
<tr>
<td>application</td>
<td>ApplicationName</td>
</tr>
<tr>
<td>destinationPayloadHex</td>
<td>UTF8</td>
</tr>
<tr>
<td>firstPacketDate</td>
<td>DateFormat</td>
</tr>
<tr>
<td>eventProcessorId</td>
<td>ProcessorName</td>
</tr>
</tbody>
</table>

This following Ariel database fields 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
Overview of Ariel Query Language

Use AQL to extract, filter, and perform actions on event and flow data that you extract from the Ariel database in IBM Security QRadar. You can use AQL to get data that might not be easily accessible from the user interface.

The following diagram shows the flow of an AQL query.
**Structure of an AQL statement**

Use the SELECT statement to select fields from events or flows in the Ariel database, which are displayed as columns. For example, the following query returns the results that are shown in the following table:

```
SELECT sourceip, destinationip, username, protocolid, eventcount FROM events
```

<table>
<thead>
<tr>
<th>sourceip</th>
<th>destinationip</th>
<th>Username</th>
<th>protocolid</th>
<th>eventcount</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.0.2.21</td>
<td>198.51.100.21</td>
<td>Joe Ariel</td>
<td>233</td>
<td>1</td>
</tr>
<tr>
<td>192.0.2.22</td>
<td>198.51.100.24</td>
<td>Jim Ariel</td>
<td>233</td>
<td>1</td>
</tr>
</tbody>
</table>

AQL queries begin with a SELECT statement to select event or flow data from the Ariel database. You can refine the data output of the SELECT statement by using the WHERE, GROUP BY, HAVING, ORDER BY, LIMIT, and LAST clauses.

**SELECT**

Use the SELECT statement to select fields from events or flows. For example, select all fields from events or flows by typing:

```
SELECT * FROM events, or SELECT * FROM flows
```
Use the following clauses to filter and manipulate the data that is returned by the SELECT statement:

**WHERE**

Use the WHERE clause to insert a condition that filters the output, for example, WHERE logsourceid='65'.

**GROUP BY**

Use the GROUP BY clause to group the results by one or more columns that you specify in the query, for example, GROUP BY logsourceid.

**HAVING**

Use the HAVING clause to specify a condition after the GROUP BY clause, for example, HAVING MAG > 3.

**ORDER BY**

Use the ORDER BY clause to order the results for a column in the AQL query in an ascending or descending order, for example, ORDER BY username DESC.

**LIMIT**

Use a LIMIT clause to limit the number of results that are returned to a specific number, for example LIMIT 50 to limit the output to 50 results.

**LAST**

Use a LAST clause to specify a time frame for the query, for example LAST 1 HOURS.

The following example incorporates all of the clauses that are described in the list:

```
SELECT sourceip, destinationip, username
FROM events
WHERE username = 'test name'
GROUP BY sourceip, destinationip
ORDER BY sourceip DESC
LIMIT 10
LAST 2 DAYS
```

**Best practices for using quotation marks in AQL queries**

In an AQL query, query terms and queried columns sometimes require single or double quotation marks so that QRadar can parse the query.

The following table defines when to use single or double quotation marks.

<table>
<thead>
<tr>
<th>Type of quotation marks</th>
<th>When to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>To specify any American National Standards Institute (ANSI) VARCHAR string to SQL such as parameters for a LIKE or equals (=) operator, or any operator that expects a VARCHAR string.</td>
</tr>
<tr>
<td></td>
<td>Examples:</td>
</tr>
<tr>
<td></td>
<td>SELECT * from events WHERE sourceip = '192.0.2.0'</td>
</tr>
<tr>
<td></td>
<td>SELECT * from events WHERE userName LIKE '%james%'</td>
</tr>
<tr>
<td></td>
<td>SELECT * from events WHERE userName = 'james'</td>
</tr>
<tr>
<td></td>
<td>SELECT * FROM events WHERE INCIDR('10.45.225.14', sourceip)</td>
</tr>
<tr>
<td></td>
<td>SELECT * from events WHERE TEXT SEARCH 'my search term'</td>
</tr>
</tbody>
</table>
Table 3. Type of quotation marks to use in a query (continued)

<table>
<thead>
<tr>
<th>Type of quotation marks</th>
<th>When to use</th>
</tr>
</thead>
</table>
| Double                  | Use double quotation marks for the following query items to specify table and column names that contain spaces or non-ASCII characters, and to specify custom property names that contain spaces or non-ASCII characters. **Examples:**
|                          | SELECT "username column" AS 'User name' FROM events
|                          | SELECT "My custom property name" AS 'My new alias' FROM events |
|                          | Use double quotation marks to define the name of a system object such as field, function, database, or an existing alias. **Example:**
|                          | SELECT "Application Category", sourceIP, EventCount AS 'Count of Events'
|                          | FROM events GROUP BY "Count of Events"
|                          | Use double quotation marks to specify an existing alias that has a space when you use a WHERE, GROUP BY, or ORDER BY clause. **Examples:**
|                          | SELECT sourceIP, destinationIP, sourcePort, EventCount AS 'Event Count', category, hasidentity, username, payload, Utf8(payload), QiD, QiDnAmE(qid) FROM events
|                          | WHERE (NOT (sourcePort <= 3003 OR hasidentity = 'True'))
|                          | AND (qid = 5000023 OR qid = 5000193)
|                          | AND (INCIDR(‘192.0.2.0/4’, sourceIP)
|                          | OR NOT INCIDR(‘192.0.2.0/4’, sourceIP)) ORDER BY "Event Count"
|                          | DESC LAST 60 MINUTES
|                          | SELECT sourceIP, destinationIP, sourcePort, EventCount AS 'Event Count', category, hasidentity, username, payload, Utf8(payload), QiD, QiDnAmE(qid)
|                          | FROM events GROUP BY "Source IP Address" DESC LAST 60 MINUTES |
| Single or double         | Use single quotation marks to specify an alias for a column definition in a query. **Example:**
|                          | SELECT username AS 'Name of User', sourceip AS 'IP Source' FROM events |
|                          | Use double quotation marks to specify an existing alias with a space when you use a WHERE, GROUP BY, or ORDER BY clause. **Example:**
|                          | SELECT sourceIP AS 'Source IP Address', EventCount AS 'Event Count', QiD, QiDnAmE(qid)
|                          | FROM events
|                          | GROUP BY "Source IP Address"
|                          | DESC LAST 60 MINUTES |

**Copying query examples from the AQL guide**

If you copy and paste a query example that contains single or double quotation marks from the AQL Guide, you must retype the quotation marks to be sure that the query parses.

**SELECT statement**

Use the SELECT statement to define the criteria that you use to retrieve event or flow data.
Use the SELECT statement to define the columns (fields) that you want to output from your query. You can use the SELECT statement to output data from an AQL function by using a column alias. Typically, you refer to events or flows in your SELECT statement but you can also use the SELECT statement with the GLOBALVIEW database, or any other database that you might have access to.

Use the SELECT statement to select the columns that you want to display in the query output.

A SELECT statement can include the following elements:

- Fields from the events or flows databases
- Custom properties from the events or flows databases
- Functions that you use with fields to represent specific data that you want to return.
  
  For example, the function ASSETHOSTNAME(sourceip) searches for the host name of an asset by source IP address at a specific time.

Use an asterisk (*) to denote all columns.

Field names and SELECT and FROM statements are not case-sensitive. For example, the following query uses different cases and it parses.

```
select Sourceip, DATEFORMAT(startTime,'YYYY-MM-dd HH:mm') as startTime from events WHERE username is not Null GROUP BY sourceip ORDER BY startTime LAST 3 hours
```

The following examples are queries that use SELECT statements:

- **SELECT * FROM flows**
  
  Returns all columns from the flows database.

- **SELECT sourceip, destinationip FROM events**
  
  Returns only the sourceip and destinationip columns from the events database.

- **SELECT sourceip, * FROM flows**
  
  Returns the sourceip column first, which is followed by all columns from the flows database.

- **SELECT sourceip AS 'MY Source IPs', FROM events**
  
  Returns the sourceip column as the alias or renamed column 'MY Source IPs'.

- **SELECT ASSETHOSTNAME(sourceip) AS 'Host Name', sourceip FROM events**
  
  Returns the output of the function ASSETHOSTNAME as the column name Host Name, and the sourceip column from the events database.

**WHERE clause**

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

You can apply the WHERE clause to add a condition to search criteria in AQL queries, which filters the search results.

A search condition 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.

You can apply the following filters when you use WHERE clause in a query:

- Equal sign (=)
- Not equal to symbol (<>)
- Less than symbol (<)
- Greater than symbol (>
• Less than or equal to symbol (\(\leq\))
• Greater than or equal to symbol (\(\geq\))
• BETWEEN between two values, for example (64 AND 512)
• LIKE case sensitive match
• ILIKE case insensitive match
• IS NULL is empty
• AND / OR combine conditions or either condition
• TEXT SEARCH text string match

Examples of WHERE clauses

The following query example shows events that have a severity level of greater than nine and are from a specific category.

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

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

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

Return events from the events database where the text 'typot' is found.

```sql
SELECT QIDNAME(qid) AS EventName,
* FROM events
WHERE TEXT SEARCH 'typot'
```

The following query outputs events from the events database where health is included in the log source name.

```sql
SELECT logsourceid, LOGSOURCEGROUPNAME(logsourceid), LOGSOURCENAME(logsourceid)
FROM events
WHERE LOGSOURCENAME(logsourceid) ILIKE '%health%%'
```

The following query outputs events where the device type ID is equal to 11 (Linux Server DSM), and where the QID is equal to 44250002, which is the identifier for Cron Status.

```sql
SELECT * FROM events
WHERE deviceType = '11'
AND qid = '44250002'
```

GROUP BY clause

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

When you use the GROUP BY clause with a column name or AQL function, only the first value is returned for the GROUP BY column, by default, even though other values might exist.
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.

```
<table>
<thead>
<tr>
<th>sourceIP</th>
<th>SUM_sourceBytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.0.2.0</td>
<td>4282590.0</td>
</tr>
<tr>
<td>10.105.2.10</td>
<td>4902509.0</td>
</tr>
<tr>
<td>10.103.70.243</td>
<td>2802715.0</td>
</tr>
<tr>
<td>10.103.77.143</td>
<td>3313370.0</td>
</tr>
<tr>
<td>10.105.32.29</td>
<td>2467183.0</td>
</tr>
<tr>
<td>10.105.96.148</td>
<td>8325356.0</td>
</tr>
<tr>
<td>10.103.73.206</td>
<td>1629768.0</td>
</tr>
</tbody>
</table>
```

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:

```
<table>
<thead>
<tr>
<th>sourceIP</th>
<th>sourceBytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.0.2.0</td>
<td>1448629</td>
</tr>
<tr>
<td>10.105.2.10</td>
<td>2412426</td>
</tr>
<tr>
<td>10.103.70.243</td>
<td>1793095</td>
</tr>
<tr>
<td>10.103.77.143</td>
<td>1449148</td>
</tr>
<tr>
<td>10.105.32.29</td>
<td>1097523</td>
</tr>
<tr>
<td>10.105.96.148</td>
<td>4096834</td>
</tr>
</tbody>
</table>
```

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), PROTOCOLNAME(protocolid)
FROM events
GROUP BY sourceIP
```

The output displays the following results:

```
<table>
<thead>
<tr>
<th>sourceIP</th>
<th>protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.0.2.0</td>
<td>TCP.tcp.ip</td>
</tr>
<tr>
<td>10.105.2.10</td>
<td>UDP.udp.ip</td>
</tr>
<tr>
<td>10.103.70.243</td>
<td>UDP.udp.ip</td>
</tr>
<tr>
<td>10.103.77.143</td>
<td>UDP.udp.ip</td>
</tr>
<tr>
<td>10.105.32.29</td>
<td>TCP.tcp.ip</td>
</tr>
<tr>
<td>10.105.96.148</td>
<td>TCP.tcp.ip</td>
</tr>
<tr>
<td>192.0.2.0</td>
<td>TCP.tcp.ip</td>
</tr>
<tr>
<td>10.105.2.10</td>
<td>ICMP.icmp.ip</td>
</tr>
</tbody>
</table>
```
HAVING clause

Use the HAVING clause in a query to apply more filters to specific data by applying filters to the results after the GROUP BY clause.

The HAVING clause follows the GROUP BY clause.

You can apply the following filters when you use a HAVING clause in a query:

- Equal sign (=)
- Not equal to symbol (<>)
- Less than symbol (<)
- Greater than symbol (>)
- Less than or equal to symbol (<=)
- Greater than or equal to symbol (>=)
- BETWEEN between two values, for example (64 AND 512)
- LIKE case-sensitive match
- ILIKE case insensitive match
- SUM/AVG total or average values
- MAX/MIN maximum or minimum values

Examples of HAVING clauses

The following query example shows results for users who triggered VPN events from more than four IP addresses (HAVING 'Count of Source IPs' > 4) in the last 24 hours.

```sql
SELECT username, UNIQUECOUNT(sourceip) AS 'Count of Source IPs'
FROM events
WHERE LOGSOURCENAME(logsourceid) ILIKE '%vpn'
AND username IS NOT NULL
GROUP BY username
HAVING "Count of Source IPs" > 4
LAST 24 HOURS
```

**Note:** When you type an AQL query, use single quotation marks for a string comparison, and use double quotation marks for a property value comparison.

The following query example shows results for events where the credibility (HAVING credibility > 5) is greater than five.

```sql
SELECT username, sourceip, credibility
FROM events
GROUP BY sourceip
HAVING credibility > 5
LAST 1 HOURS
```

The following query groups results by source IP but displays only results where the magnitude (HAVING magnitude > 5) is greater than five.

```sql
SELECT sourceIP, magnitude
FROM events
GROUP BY sourceIP
HAVING magnitude > 5
```

ORDER BY clause

Use the ORDER BY clause to sort the resulting view that is based on expression results. The result is sorted by ascending or descending order.
**Note:** When you type an AQL query, use single quotation marks for a string comparison, and use double quotation marks for a property value comparison.

You can use the ORDER BY clause on one or more columns.

Use the GROUP BY and ORDER BY clauses in a single query.

Sort in ascending or descending order by appending the ASC or DESC keyword to the ORDER BY clause.

**Examples of ORDER BY clauses**

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

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

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

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

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. For example, the following query displays the most traffic intensive IP address in descending order:

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

**Note:**

When you use the GROUP BY clause with a column name or AQL function, only the first value is returned for the GROUP BY column, by default, even though other values might exist.

**LIKE clause**

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

You can search fields by using the LIKE clause.

The following table shows the wildcard options are supported by the Ariel Query Language (AQL).

<table>
<thead>
<tr>
<th>Wildcard character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>Matches a string of zero or more characters</td>
</tr>
<tr>
<td>_</td>
<td>Matches any single character</td>
</tr>
</tbody>
</table>

**Examples of LIKE clauses**

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

```sql
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.

**Examples of the Count function**

The following query returns the count of all events with credibility that is greater than or equal to 9.

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

The following query returns the count of assets by location and source IP address.

```
SELECT ASSETPROPERTY('Location',sourceip)
AS location, COUNT(*)
FROM events
GROUP BY location
LAST 1 days
```

The following query returns the user names, source IP addresses, and count of events.

```
SELECT username, sourceip,
COUNT(*) FROM events
GROUP BY username
LAST 600 minutes
```

The sourceip column is returned as FIRST_sourceip.

One sourceip is returned only per username, even if another sourceip exists.

**Note:**

When you use the GROUP BY clause with a column name or AQL function, only the first value is returned for the GROUP BY column, by default, even though other values might exist.
AQL logical and comparison operators

Operators are used in AQL statements to determine any equality or difference between values. By using operators in the WHERE clause of an AQL statement, the results are filtered by those results that match the conditions in the WHERE clause.

The following table lists the supported logical and comparison operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Multiplies two values and returns the result.</td>
<td>SELECT * FROM flows WHERE sourceBytes * 1024 &lt; 1</td>
</tr>
<tr>
<td>=</td>
<td>The equal to operator compares two values and returns true if they are equal.</td>
<td>SELECT * FROM EVENTS WHERE sourceIP = destinationIP</td>
</tr>
<tr>
<td>!=</td>
<td>Compares two values and returns true if they are unequal.</td>
<td>SELECT * FROM events WHERE sourceIP != destinationIP</td>
</tr>
<tr>
<td>&lt; AND &lt;=</td>
<td>Compares two values and returns true if the value on the left side is less than or equal to the value on the right side.</td>
<td>SELECT * FROM flows WHERE sourceBytes &lt; 64 AND destinationBytes &lt;= 64</td>
</tr>
<tr>
<td>&gt; AND &gt;=</td>
<td>Compares two values and returns true if the value on the left side is greater than or equal to the value on the right side.</td>
<td>SELECT * FROM flows WHERE sourceBytes &gt; 64 AND destinationBytes &gt;= 64</td>
</tr>
<tr>
<td>/</td>
<td>Divides two values and returns the result.</td>
<td>SELECT * FROM flows WHERE sourceBytes / 8 &gt; 64</td>
</tr>
<tr>
<td>+</td>
<td>Adds two values and returns the result.</td>
<td>SELECT * FROM flows WHERE sourceBytes + destinationBytes &lt; 64</td>
</tr>
<tr>
<td>-</td>
<td>Subtracts one value from another and returns the result.</td>
<td>SELECT * FROM flows WHERE sourceBytes - destinationBytes &lt; 64</td>
</tr>
<tr>
<td>^</td>
<td>Takes a value and raises it to the specified power and returns the result.</td>
<td>SELECT * FROM flows WHERE sourceBytes ^ 2 &lt; 256</td>
</tr>
<tr>
<td>%</td>
<td>Takes the modulo of a value and returns the result.</td>
<td>SELECT * FROM flows WHERE sourceBytes % 8 == 7</td>
</tr>
<tr>
<td>AND</td>
<td>Takes the left side and right side of a statement and returns true if both are true.</td>
<td>SELECT * FROM events WHERE (sourceIP = destinationIP) AND (sourcePort = destinationPort)</td>
</tr>
<tr>
<td>BETWEEN</td>
<td>Takes in a left side and two values and returns true if the left side is between the two values.</td>
<td>SELECT * FROM events WHERE magnitude BETWEEN 1 AND 5</td>
</tr>
<tr>
<td>(X,Y)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operator</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>COLLATE</td>
<td>Parameter to order by that allows a BCP47 language tag to collate.</td>
<td>SELECT * FROM EVENTS ORDER BY sourceIP DESC COLLATE 'de-CH'</td>
</tr>
<tr>
<td>IN</td>
<td>Specifies multiple values in a WHERE clause. The IN operator is a shorthand for multiple OR conditions.</td>
<td>SELECT * FROM EVENTS WHERE SourceIP IN ('192.0.2.1', '::1', '198.51.100.0')</td>
</tr>
<tr>
<td>INTO</td>
<td>Creates a named cursor that contains results that can be queried at a different time.</td>
<td>SELECT * FROM EVENTS INTO 'MyCursor' WHERE....</td>
</tr>
<tr>
<td>NOT</td>
<td>Takes in a statement and returns true if the statement evaluates as false.</td>
<td>SELECT * FROM EVENTS WHERE NOT (sourceIP = destinationIP)</td>
</tr>
<tr>
<td>ILIKE</td>
<td>Matches if the string passed is LIKE the passed value and is not case sensitive. Use % as a wildcard.</td>
<td>SELECT * FROM events WHERE userName ILIKE '%bob%'</td>
</tr>
<tr>
<td>IMATCHES</td>
<td>Matches if the string matches the provided regular expression and is not case sensitive.</td>
<td>SELECT * FROM events WHERE userName IMATCHES '^.<em>bob.</em>$'</td>
</tr>
<tr>
<td>LIMIT</td>
<td>Limits the number of results to the provided number.</td>
<td>SELECT * FROM events LIMIT 100 START '2015-10-28 10:00' STOP '2015-10-28 11:00'</td>
</tr>
<tr>
<td>LIKE</td>
<td>Matches if the string passed is LIKE the passed value but is case sensitive. Use % as a wildcard.</td>
<td>SELECT * FROM events WHERE userName LIKE '%bob%'</td>
</tr>
<tr>
<td>MATCHES</td>
<td>Matches if the string matches the provided regular expression.</td>
<td>SELECT * FROM events WHERE userName MATCHES '^.<em>bob.</em>$'</td>
</tr>
<tr>
<td>NOT NULL</td>
<td>Takes in a value and returns true if the value is not null.</td>
<td>SELECT * FROM events WHERE userName IS NOT NULL</td>
</tr>
<tr>
<td>OR</td>
<td>Takes the left side of a statement and the right side of a statement and returns true if either side is true.</td>
<td>SELECT * FROM events WHERE (sourceIP = destinationIP) OR (sourcePort = destinationPort)</td>
</tr>
</tbody>
</table>
Table 5. Logical and comparison operators (continued)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT SEARCH</td>
<td>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 get a syntax error. Place TEXT SEARCH in the first position of the WHERE clause. 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 User Guide.</td>
<td>SELECT * FROM events WHERE TEXT SEARCH 'firewall' AND sourceip='192.168.1.1' 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'</td>
</tr>
</tbody>
</table>

Examples of logical and comparative operators

- To find events that are not parsed, type the following query:
  ```sql
  SELECT * FROM events
  WHERE payload = 'false'
  ```

- To find events that return an offense and have a specific source IP address, type the following query:
  ```sql
  SELECT * FROM events
  WHERE sourceIP = '192.0.2.0'
  AND hasOffense = 'true'
  ```

- To find events that include the text "firewall", type the following query:
  ```sql
  SELECT QIDNAME(qid) AS EventName, *
  FROM events
  WHERE TEXT SEARCH 'firewall'
  ```

AQL data calculation and formatting functions

Use Ariel Query Language (AQL) calculation and formatting functions on search results that are retrieved from the Ariel databases.

This list describes the AQL functions that are used for calculations and data formatting:

- “BASE64” on page 20
- “CONCAT” on page 20
- “DATEFORMAT” on page 20
- “DOUBLE” on page 20
- “LONG” on page 20
- “LOWER” on page 22
- “NOW” on page 22
- “PARSEDATETIME” on page 21
- “PARSETIMESTAMP” on page 21
- “REPLACEALL” on page 22
- “REPLACEFIRST” on page 23
BASE64

**Purpose**
Returns a Base64 encoded string that represents binary data.

**Example**
```sql
SELECT BASE64(payload)
FROM events
```
Returns the payloads for events in BASE64 format.

**CONCAT**

**Purpose**
Concatenates all passed strings into one string.

**Example**
```sql
SELECT CONCAT(username, ':', sourceip, ':', destinationip)
FROM events LIMIT 5
```

**DATEFORMAT**

**Purpose**
Formats time in milliseconds since 00:00:00 Coordinated Universal Time (UTC) on January 1, 1970 to a user-readable form.

**Examples**
```sql
SELECT DATEFORMAT(startTime, 'yyyy-MM-dd hh:mm:ss')
AS Start_Time
FROM events

SELECT DATEFORMAT(starttime, 'yyyy-MM-dd hh:mm')
AS 'Start Time',
DATEFORMAT(endtime, 'yyyy-MM-dd hh:mm')
AS End_time,
QIDDESCRIPTION(qid)
AS 'Event Name'
FROM events
```

**DOUBLE**

**Purpose**
Converts a value that represents a number into a double.

**Example**
```sql
DOUBLE('1234')
```

**LONG**

**Purpose**
Converts a value that represents a number into a long integer.
Examples

The example returns the destination IP address, and the sum of the source and destination bytes in the TotalBytes column.

SELECT
LONG(sourceip)
AS long_ip
FROM events
INTO <cursor_name>
WHERE (long_ip & 0xff) = 0x<hexadecimal value of IP address>000000
GROUP BY long_ip
LIMIT 20

In QRadar 7.3.1, you can use the LONG function to convert IP addresses into a long integer. QRadar uses long integers with bitwise operators to do IP address arithmetic and filtering in AQL queries. In the example, the source IP is returned as an integer, which is used by the bitwise AND operator.

In the example, the <ff> corresponds with <hexadecimal value of IP address>, which is in the first octet position for an IP address. The <cursor_name> can be any name that you want to use.

For example, if you want to return all source IP addresses with the number 9 in the first octet, then substitute the hexadecimal value 9, which is the same as the decimal value, in <hexadecimal value of IP address>.

See more examples of the long function that are used with bitwise operators.

PARSEDATETIME

Purpose
Pass a time value to the parser, for example, PARSEDATETIME('time reference'). The time reference indicates the parse time for the query.

Example
SELECT * FROM events
START PARSEDATETIME('1 hour ago')

See more examples of time functions

PARSETIMESTAMP

Purpose
Parse the text representation of date and time and convert it to UNIX epoch time.

For example, parse the following text date format:
Thursday, August 24, 2017 3:30:32 PM GMT +01:00 and convert it to the following epoch timestamp: 1503588632.

This function makes it easier to issue calls from the API that are based on scripts.

Example of how the time format conversion works

The following example demonstrates how the DATETIME function converts epoch time to a text timestamp by using the specified date format, and then the PARSETIMESTAMP function is used to convert the text timestamp to an epoch time format.
SELECT startime, DATEFORMAT(starttime, 'EEE, MMM d, "yyyy"') AS "text time format",
      PARSETIMESTAMP('EEE, MMM d, "yyyy"', "text time format") AS 'epoch time returned' FROM events limit 5

The following example displays an extract of the output from the query:

```
    starttime  text time format  epoch time returned
    1503920389888  Mon, 08 28, "2017"  1503920389888
```

**Example of how PARSETIMESTAMP might be used to convert times to epoch time so that time calculations can be made.**

In the following example, events are returned when the time difference between logout and login times is less that 1 hour.

The `EEE, d MMM yyyy HH:mm:ss.SSSZ` time format is just one example of a time format that you might use, and `my_login` and `my_logout` are custom properties in a known time format, for example, `EEE, MMM dd`, "yy".

```
SELECT *
FROM events
WHERE PARSETIMESTAMP('EEE, d MMM yyyy HH:mm:ss.SSSZ', my_logout) - PARSETIMESTAMP('EEE, d MMM yyyy HH:mm:ss.SSSZ', my_login) < 3600000
```

See more examples of time functions

---

**NOW**

**Purpose**

Returns the current time that is expressed as milliseconds since the time 00:00:00 Coordinated Universal Time (UTC) on January 1, 1970.

**Example**

```
SELECT ASSETUSER(sourceip, NOW())
FROM events
```

Find the user of the asset at this moment in time (NOW).

---

**LOWER**

**Purpose**

Returns an all lowercase representation of a string.

**Example**

```
SELECT LOWER(username),
      LOWER(LOGSOURCENAME(logsourceid))
FROM events
```

Returns user names and log source names in lowercase.

---

**REPLACEALL**

**Purpose**

Match a regex and replace all matches with text.

Replaces every subsequence (`arg2`) of the input sequence that matches the pattern (`arg1`) with the replacement string (`arg3`).

**Example**

```
REPLACEALL(\d{16},
        username, 'censored')
```
REPLACEFIRST

Purpose
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 replacement string (arg3).

Example
REPLACEFIRST('\d{16}', username, 'censored')

STR

Purpose
Converts any parameter to a string.

Example
STR(sourceIP)

STRLEN

Purpose
Returns the length of this string.

Example
SELECT STRLEN(sourceIP), STRLEN(username) from events

Returns the string length for sourceip and username.

STRPOS

Purpose
Returns the position (index - starts at zero) of a string in another string. Searches in string for the index of the specified substring. You can optionally specify an extra parameter to indicate at what position (index) to start looking for the specified pattern.

The search for the string starts at the specified offset and moves towards the end of string.

STRPOS(string, substring, index)
Returns -1 if the substring isn't found.

Examples
SELECT STRPOS(username, 'name') FROM events
SELECT STRPOS(sourceip, '180', 2) FROM events

SUBSTRING

Purpose
Copies a range of characters into a new string.

Examples
SELECT SUBSTRING(userName, 0, 3) FROM events
SELECT SUBSTRING(sourceip, 3, 5) FROM events

UPPER

Purpose
Returns an all uppercase representation of a string.
Example

```
SELECT
  UPPER(username),
  UPPER(LOGSOURCENAME(logsourceid))
FROM events
```

Returns user names and log source names in uppercase.

**UTF8**

** Purpose **

Returns the UTF8 string of a byte array.

**Example**

```
SELECT UTF8(payload)
FROM events
WHERE sourceip='192.0.2.0'
```

Returns the UTF8 payload for events where the source IP address is 192.0.2.0

---

**AQL data aggregation functions**

Ariel Query Language (AQL) aggregate functions help you to aggregate and manipulate the data that you extract from the Ariel database.

**Data aggregation functions**

Use the following AQL functions to aggregate data, and to do calculations on the aggregated data that you extract from the AQL databases:

- "AVG"
- "COUNT" on page 25
- "FIRST" on page 25
- "GROUP BY" on page 25
- "HAVING" on page 25
- "LAST" on page 26
- "MIN" on page 26
- "MAX" on page 26
- "STDEV" on page 26
- "STDEVP" on page 26
- "SUM" on page 27
- "UNIQUECOUNT" on page 27

**AVG**

** Purpose **

Returns the average value of the rows in the aggregate.

**Example**

```
SELECT sourceip,
  AVG(magnitude)
FROM events
GROUP BY sourceip
```
COUNT

Purpose
Returns the count of the rows in the aggregate.

Example
```
SELECT sourceip, COUNT(*)
FROM events
GROUP BY sourceip
```

FIRST

Purpose
Returns the first entry of the rows in the aggregate.

Example
```
SELECT sourceip, FIRST(magnitude)
FROM events
GROUP BY sourceip
```

GROUP BY

Purpose
Creates an aggregate on one or more columns.

When you use the GROUP BY clause with a column name or AQL function, only the first value is returned for the GROUP BY column, by default, even though other values might exist.

To return values other than the default first value, use functions such as COUNT, MAX, AVG.

Examples
```
SELECT sourceip, COUNT(*)
FROM events
GROUP BY sourceip, destinationip

SELECT username, sourceip, COUNT(*)
FROM events
GROUP BY username
LAST 5 minutes

SELECT username, COUNT(sourceip), COUNT(*)
FROM events
GROUP BY username
LAST 5 minutes
```

The sourceip column is returned as FIRST_sourceip. Only one sourceip is returned per username, even if another sourceip exists.

The sourceip column is returned as COUNT_sourceip. The count for sourceip results is returned per username.

HAVING

Purpose
Uses operators on the result of a grouped by column.

Example
SELECT sourceip, 
MAX(magnitude) 
AS MAG 
FROM events 
GROUP BY sourceip 
HAVING MAG > 5

See more examples

Saved searches that include the having clause and that are used for scheduled reports or time-series graphs are not supported.

LAST
Purpose
Returns the last entry of the rows in the aggregate.

Example
SELECT sourceip, 
LAST(magnitude) 
FROM events 
GROUP BY sourceip

MIN
Purpose
Returns the minimum value of the rows in the aggregate.

Example
SELECT sourceip, 
MIN(magnitude) 
FROM events 
GROUP BY sourceip

MAX
Purpose
Returns the maximum value of the rows in the aggregate.

Example
SELECT sourceip, 
MAX(magnitude) 
FROM events 
GROUP BY sourceip

STDEV
Purpose
Returns the Sample Standard Deviation value of the rows in the aggregate.

Example
SELECT sourceip, 
STDEV(magnitude) 
FROM events 
GROUP BY sourceip

STDEVP
Purpose
Returns the Population Standard Deviation value of the rows in the aggregate.

Example
SELECT sourceip, 
STDEV(magnitude) 
FROM events 
GROUP BY sourceip

SUM
Purpose
Returns the sum of the rows in the aggregate.

Example
SELECT sourceip, 
SUM(sourceBytes) 
FROM flows 
GROUP BY sourceip

UNIQUECOUNT
Purpose
Returns the unique count of the value in the aggregate.

Example
SELECT username, 
UNIQUECOUNT(sourceip) 
AS CountSrcIP 
FROM events 
GROUP BY sourceip

AQL data retrieval functions

Use the Ariel Query Language (AQL) built-in functions to retrieve data by using data query functions and field ID properties from the Ariel database.

Use the following AQL functions to extract data from the Ariel databases:

Data retrieval functions
• “APPLICATIONNAME” on page 28
• “ARIELSERVERS4EPID” on page 28
• “ARIELSERVERS4EPNAME” on page 29
• “ASSETHOSTNAME” on page 29
• “ASSETPROPERTY” on page 30
• “ASSETUSER” on page 30
• “CATEGORYNAME” on page 30
• “COMPONENTID” on page 31
• “DOMAINNAME” on page 31
• “GLOBALVIEW” on page 31
• “GEO::LOOKUP” on page 32
• “GEO::DISTANCE” on page 32
• “HOSTNAME” on page 32
• “INCIDR” on page 32
• “INOFFENSE” on page 33
• “LOGSOURCENAME” on page 33
• “LOGSOURCEGROUPNAME” on page 33
• “LOGSOURCETYPENAME” on page 33
APPLICATIONNAME

Purpose

Returns flow application names by application ID

Parameters

Application ID

Example

```
SELECT APPLICATIONNAME(applicationid)
AS 'Name of App'
FROM flows
```

Returns the names of applications from the flows database. These application names are listed in the Name of App column, which is an alias.

ARIELSERVERS4EPID

Purpose

Use the ARIELSERVERS4EPID function to specify the Event Processor ID when you use it with PARAMETERS REMOTESERVERS or PARAMETERS EXCLUDESERVERS.

Parameters

ARIELSERVERS4EPID(processor_ID)

The following examples show how to use the ARIELSERVERS4EPID function with PARAMETERS REMOTESERVERS or PARAMETERS EXCLUDESERVERS:

```
PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPID(processor_ID)
PARAMETERS REMOTESERVERS=ARIELSERVERS4EPID(processor_ID)
```

Examples

In the following example, only the search results from ARIELSERVERS4EPID(8) are included in the output. If the processor ID that you specify as a parameter for the ARIELSERVERS4EPID function is not in your QRadar deployment, then the query does not run.

```
SELECT ARIELSERVERS4EPID(8), ARIELSERVERS4EPID(11), processorid,
PROCESSORNAME(processorid),
LOGSOURCENAME(logsourceid) from events
GROUP BY logsourceid
LAST 20 MINUTES
PARAMETERS REMOTESERVERS=ARIELSERVERS4EPID(8)
```
You can also use the ARIELSERVERS4EPID function to return the Ariel servers that are connected to a specific Event Processor that is identified by ID, as shown in the following example:

```
SELECT processorid, PROCESSORNAME(processorid),
      ARIELSERVERS4EPID(processorid)
FROM events GROUP BY processorid
```

### ARIELSERVERS4EPNAME

**Purpose**

You use the ARIELSERVERS4EPNAME function to specify the Event Processor name when you use it with PARAMETERS REMOTESERVERS or PARAMETERS EXCLUDESERVERS.

**Parameters**

```
ARIELSERVERS4EPNAME('eventprocessor_name')
```

The following examples show how you use ARIELSERVERS4EPNAME PARAMETERS REMOTESERVERS or PARAMETERS EXCLUDESERVERS:

```
PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPNAME ('eventprocessor104')
PARAMETERS REMOTESERVERS=ARIELSERVERS4EPNAME ('eventprocessor255')
```

**Examples**

In the following example, records from servers that are associated with eventprocessor104 are excluded from the search.

```
SELECT processorid, PROCESSORNAME(processorid),
      LOGSOURCENAME(logsourceid)
FROM events
GROUP BY logsourceid
PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPNAME ('eventprocessor104')
```

You can also use the function to return Ariel servers that are associated with an Event Processor that is identified by name.

```
SELECT PROCESSORNAME(processorid),
      ARIELSERVERS4EPNAME(PROCESSORNAME(processorid))
FROM events GROUP BY processorid
```

Returns Ariel servers for the named Event Processor.

### ASSETHOSTNAME

**Purpose**

Searches for the host name of an asset at a point in time.

The domain can optionally be specified to target an asset on a particular domain.

```
ASSETHOSTNAME(sourceip)
ASSETHOSTNAME(sourceip, NOW())
ASSETHOSTNAME(sourceip, domainid)
```

**Parameters**

- IP address, (timestamp and domain ID are optional)

If the time stamp is not specified, the current time is used.

**Examples**

```
SELECT ASSETHOSTNAME(destinationip, NOW())
AS 'Host Name'
FROM events
```
SELECT ASSETHOSTNAME(sourceip, NOW())
AS 'Host Name'
FROM events

Returns the host name of the asset at the time of the query.

ASSETPROPERTY

Purpose

Looks up a property for an asset.

The domain can optionally be specified to target an asset on a particular domain.

ASSETPROPERTY
('Unified Name', sourceIP, domainId)

Parameters

Property name, IP address
Domain ID is optional

Example

SELECT ASSETPROPERTY('Location', sourceip)
AS Asset_location,
COUNT(*)
AS 'event count'
FROM events
GROUP BY Asset_location
LAST 1 days

Returns the asset location that is affiliated with the source IP address.

ASSETUSER

Purpose

Searches for the user of an asset at a point in time.

Domain can optionally be specified to target an asset in a specific domain.

ASSETUSER(sourceIP, NOW(), domainId)

Parameters

IP address, (timestamp and domain ID are optional)
If the time stamp is not specified, the current time is used.

Example

SELECT ASSETUSER(sourceip, now())
AS 'Username of Asset'
FROM events

Returns the user name that is affiliated with the source IP address.

CATEGORYNAME

Purpose

Searches for the name of a category by the category ID.

CATEGORYNAME(Category)

Parameters

Category
Example

```sql
SELECT sourceip, category, CATEGORYNAME(category) AS 'Category name' FROM events
```

Returns the source IP, category ID, and category name

**COMPONENTID**

**Purpose**

Retrieves the ID for a component with a given name.

For example, `ARIELSERVERS4EPNAME()` is a shortcut for the `ARIELSERVERS4EPID(COMPONENTID(<event_processor_name>))` function.

**Parameters**

`COMPONENTID(<component_name>)`

**Example**

```sql
SELECT * FROM events WHERE processorid = COMPONENTID('eventprocessor0')
```

Retrieves events for the named Event Processor.

**DOMAINNAME**

**Purpose**

Searches for the domain name by the domain ID.

`DOMAINNAME(domainID)`

**Parameters**

Domain ID

**Example**

```sql
SELECT sourceip, username, DOMAINNAME(domainid) AS 'Domain name' FROM events
```

Returns source IP, user name, and domain names from events database

**GLOBALVIEW**

**Purpose**

Returns the GLOBALVIEW database results for a given saved search name based on the time range that is input.

This query can be run only by using API.

For more information about accessing a GLOBALVIEW database, see the *IBM Security QRadar Administration Guide*.

**Parameters**

Saved search, time range (DAILY, NORMAL, HOURLY)

**Example**

```sql
SELECT * FROM GLOBALVIEW ('Top Log Sources', 'DAILY')
```

LAST 2 days
**GEO::LOOKUP**

**Purpose**

Returns location data, provided by MaxMind, for a selected IP address.

**Parameters**

- IP address (required)
- Strings (at least one required):
  - city, continent, physical_country, registered_country, represented_country, location, postal, subdivisions, traits, geo_json

**Example**

```sql
SELECT sourceip, GEO::LOOKUP(sourceip, 'city')
AS GEO_CITY
FROM events last 10 minutes
```

**GEO::DISTANCE**

**Purpose**

Returns the distance, in kilometers, of two IP addresses.

**Parameters**

- IP address (two required)

**Example**

```sql
SELECT GEO::DISTANCE(sourceip, destinationip)
AS GEO_DISTANCE
FROM events last 10 minutes
```

**HOSTNAME**

**Purpose**

Returns the host name of an event processor with a certain processorID.

**Parameters**

- Processor ID

**Example**

```sql
SELECT HOSTNAME(processorId) FROM events
```
**INOFFENSE**

Purpose

If an event or flow belongs to the specified offense, it returns true.

Parameters

- Offense ID

Example

```sql
SELECT * FROM events
WHERE InOffense(123)
SELECT * FROM flows
WHERE InOffense(123)
```

**LOGSOURCECNAME**

Purpose

Looks up the name of a log source by its log source ID.

LOGSOURCECNAME(logsourceid)

Parameters

- Log source ID

Example

```sql
SELECT * FROM events
WHERE LOGSOURCECNAME(logsourceid) ILIKE '%mylogsourcename%'

SELECT LOGSOURCECNAME(logsourceid) AS Log_Source
FROM events
```

Returns only results that include mylogsourcename in their log source name.

**LOGSOURCEGROUPNAME**

Purpose

Searches for the name of a log source group by its log source group ID.

LOGSOURCEGROUPNAME(deviceGroupList)

Parameters

- Device group list

Example

```sql
SELECT sourceip, logsourceid
FROM events
WHERE LOGSOURCEGROUPNAME(devicegrouplist) ILIKE '%other%'

SELECT LOGSOURCEGROUPNAME(devicegrouplist) AS Log_Source
FROM events
```

Returns the source IP address and log source IDs for log source groups that have 'other' in their name.

**LOGSOURCECYPENAME**

Purpose

Searches for the name of a log source type by its device type.

LOGSOURCECYPENAME(deviceType)
Parameters

Device type

Example

```sql
SELECT LOGSOURCETYPENAME(devicetype)
AS 'Device names', COUNT(*)
FROM events
GROUP BY "Device names"
LAST 1 DAYS
```

Returns device names and the event count.

**All log sources functions example:**

```sql
SELECT logsourceid,
LOGSOURCENAME(logsourceid)
AS 'Name of log source',
LOGSOURCEGROUPNAME(devicegrouplist)
AS 'Group Names',
LOGSOURCETYPENAME(devicetype)
AS 'Devices'
FROM events
GROUP BY logsourceid
```

Returns log source names, log source group names, and log source device names.
When you use the GROUP BY function, the first item only in the GROUP BY list is shown in the results.

**MATCHESASSETSEARCH**

**Purpose**

If the asset is returned in the results of the saved search, it returns true.

```sql
MATCHESASSETSEARCH
('My Saved Search', sourceIP)
```

**Parameters**

Saved Search Name, IP address

**Example**

```sql
MATCHESASSETSEARCH
('My Saved Search', sourceIP)
```

**NETWORKNAME**

**Purpose**

Searches for the network name from the network hierarchy for the host that is passed in.

```sql
NETWORKNAME(sourceip)
```

The domain can optionally be specified to target a network in a particular domain.

```sql
NETWORKNAME(sourceip, domainId)
```

**Parameters**

Host property (domain is optional)

**Examples**

```sql
SELECT NETWORKNAME(sourceip)
LIKE 'servers'
AS 'My Networks'
FROM flows
```

Returns any networks that have the name servers.
SELECT NETWORKNAME(sourceip, domainID) ILIKE 'servers' AS 'My Networks' FROM flows

Returns any networks that have the name servers in a specific domain.

SELECT NETWORKNAME(sourceip) AS 'Src Net', NETWORKNAME(destinationip) AS Dest_net FROM events

Returns the network name that is associated with the source and destination IP addresses.

PARAMETERS EXCLUDESERVERS

Purpose
Filters search criteria by excluding the specified servers.

Parameters

[Server IP address:Port number]

Use port 32006 for an Event Processor, and port 32011 for a Console.

Parameters accept a comma-separated list of arguments. For example, "host1:port1,host2:port2,host3:port3".

Examples

In the following example, search results from 192.0.2.0 are excluded. To exclude a Console, you must use localhost or 127.0.0.1. Do not use the IP address of the Console in this query.

SELECT processorid,PROCESSORNAME(processorid), LOGSOURCENAME(logsourceid) from events GROUP BY logsourceid PARAMETERS EXCLUDESERVERS='192.0.2.0:32006'

In the following example, search results from the Console are excluded:

SELECT processorid,PROCESSORNAME(processorid), LOGSOURCENAME(logsourceid) FROM events GROUP BY logsourceid start '2017-03-15 10:26' STOP '2017-03-15 10:30' PARAMETERS EXCLUDESERVERS='127.0.0.1:32011'

In the following example, search results from the Console are excluded. The Console is referred to as localhost in this example.

SELECT processorid,PROCESSORNAME(processorid), LOGSOURCENAME(logsourceid) from events GROUP BY logsourceid start '2017-03-15 10:25' STOP '2017-03-15 10:30' PARAMETERS EXCLUDESERVERS='localhost:32011'

The following example uses multiple arguments to exclude search results from the Console and two other servers.

SELECT processorid,PROCESSORNAME(processorid), LOGSOURCENAME(logsourceid) from events GROUP BY logsourceid start '2017-04-15 10:25' STOP '2017-04-15 10:30' PARAMETERS EXCLUDESERVERS='127.0.0.1:32011,192.0.2.0:32006,172.11.22.31:32006'
Specify the ID of the Event Processor in your query by using the following function:
PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPID(\texttt{processor\_ID})

Refine your query by using ARIELSERVERS4EPID with PARAMETERS EXCLUDESERVERS to specify the Event Processor ID that you want to exclude from your search. You can specify one or more Event Processor IDs.

**Example**

In the following example, all results from ARIELSERVERS4EPID(8) are excluded in the search.

```sql
SELECT processorid, PROCESSORNAME(processorid), LOGSOURCENAME(logsourceid) from events GROUP BY logsourceid LAST 20 MINUTES PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPID(8)
```

Specify the name of the Event Processor in your query by using the following function:
PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPNAME (\texttt{\textquoteleft processor\_name\textquoteright})

Refine your query by using ARIELSERVERS4EPNAME with PARAMETERS EXCLUDESERVERS to specify the Event Processor by name. You can specify one or more Event Processor names.

**Example**

In the following example, records from servers that are associated with eventprocessor104 are excluded from the search.

```sql
SELECT processorid, PROCESSORNAME(processorid), LOGSOURCENAME(logsourceid) FROM events GROUP BY logsourceid PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPNAME ('eventprocessor104')
```

**PARAMETERS REMOTESERVERS**

**Purpose**
Use the PARAMETERS REMOTESERVERS function to narrow your search to specific servers, which speeds up your search by not searching all hosts.

**Parameters**

[Server IP address:Port number]

Use port 32006 for an Event Processor, and port 32011 for a Console.

Use a comma-separated list for multiple arguments, for example, "host1:port1,host2:port2,host3:port3".

**Examples**

In the following example, only the specified server is searched.

```sql
SELECT * FROM EVENTS START '2016-09-08 16:42' STOP '2016-09-08 16:47' PARAMETERS REMOTESERVERS='192.0.2.0:32006'
```

In the following example, multiple servers are specified, which includes search results from the Console and two other servers.
```sql
SELECT processorid, PROCESSORNAME(processorid), LOGSOURCENAME(logsourceid) from events
GROUP BY logsourceid
start '2017-04-15 10:25'
STOP '2017-04-15 10:30'
PARAMETERS REMOTESERVERS='127.0.0.1:32011,192.0.2.0:32006,172.11.22.31:32006'
```

Specify the ID of the Event Processor in your query by using the following function:

```sql
PARAMETERS REMOTESERVERS=ARIELSERVERS4EPID(processor_ID)
```

Refine your query by using ARIELSERVERS4EPID with PARAMETERS REMOTESERVERS to specify the ID of the Event Processor that you want to include in your search. You can specify one or more Event Processor IDs.

**Example**

In the following example, only the search results from ARIELSERVERS4EPID(8) are included in the output.

```sql
SELECT ARIELSERVERS4EPID(8), ARIELSERVERS4EPID(11), processorid, PROCESSORNAME(processorid), LOGSOURCENAME(logsourceid) from events
GROUP BY logsourceid
LAST 20 MINUTES
PARAMETERS REMOTESERVERS=ARIELSERVERS4EPID(8)
```

**Note:** If the processor ID that you specify as a parameter for the ARIELSERVERS4EPID function is not in your QRadar deployment, then the query does not run.

Specify the name of the Event Processor in your query by using the following function:

```sql
PARAMETERS REMOTESERVERS=ARIELSERVERS4EPNAME ('eventprocessor_name')
```

Refine your query by using ARIELSERVERS4EPNAME and PARAMETERS REMOTESERVERS to specify the name of the Event Processor that you want to include in your search. You can specify one or more Event Processor names.

**Example**

In the following example, only search records that are associated with eventprocessor104 are included in the search results.

```sql
SELECT processorid, PROCESSORNAME(processorid), LOGSOURCENAME(logsourceid)
FROM events
GROUP BY logsourceid
PARAMETERS REMOTESERVERS=ARIELSERVERS4EPNAME ('eventprocessor104')
```

**PROCESSORNAME**

**Purpose**

Returns the name of a processor by the processor ID.

```sql
PROCESSORNAME(processorid)
```

**Parameters**

- Processor ID number

**Example**

```sql
SELECT sourceip, PROCESSORNAME(processorid) AS 'Processor Name'
FROM events
```

Returns the source IP address and processor name from the events database.

**Example**

```sql
SELECT sourceip, processorid, PROCESSORNAME(processorid) AS 'Processor Name'
FROM events
```

Returns the source IP address and processor name from the events database.
SELECT processorid, PROCESSORNAME(processorid)
FROM events WHERE processorid=104
GROUP BY processorid LAST 5 MINUTES

Returns results from the Event Processor that has a processor ID equal to 104.

**PROTOCOLNAME**

**Purpose**
Returns the name of a protocol by the protocol ID

**Parameters**
Parameter: Protocol ID number

**Example**
SELECT sourceip, PROTOCOLNAME(protocolid)
AS 'Name of protocol'
FROM events

Returns the source IP address and protocol name from the events database.

**QIDNAME**

**Purpose**
Searches for the name of a QID by its QID.

**Parameters**
Parameter: QID

**Example**
SELECT QIDNAME(qid)
AS 'My Event Names', qid
FROM events

Returns QID name and QID number.

**QIDDESCRIPTION**

**Purpose**
Searches for the QID description by its QID.

**Parameters**
Parameter: QID

**Example**
SELECT QIDDESCRIPTION(qid)
AS 'My_Event_Names', QIDNAME(qid)
AS 'QID Name'
FROM events

Returns QID description and QID name.

**REFERENCEMAP**

**Purpose**
Searches for the value for a key in a reference map.

ReferenceMap('Value',Key,domainID)
Although the domainID is optional, in a domain-enabled environment, the search is limited to only shared reference data when the domainID is excluded.

Parameters
String, String, Integer

Example
SELECT
REFERENCEMAP('Full_name_lookup', username, 5)
AS Name_of_User
FROM events

Searches for the userName (key) in the Full_name_lookup reference map in the specified domain, and returns the full name (value) for the user name (key).

REFERENCEMAPSETCONTAINS

Purpose
If a value exists for a key in a reference map of sets, for a domain, it returns true.
REFERENCEMAPSETCONTAINS(MAP_SETS_NAME, KEY, VALUE)

Parameters
String, String, String

Example
ReferenceMapSetContains('RiskyUsersForIps','sourceIP','userName')

REFERENCETABLE

Purpose
Searches for the value of a column key in a table that is identified by a table key in a specific reference table collection.
REFERENCETABLE
(testTable,'value','key', domainID)
or
REFERENCETABLE
(testTable,'value','key' domainID)

Although the domainID is optional, in a domain-enabled environment, the search is limited to only shared reference data when the domainID is excluded.

Parameters
String, String, String (or IP address), Integer

Example
SELECT
REFERENCETABLE('user_data','FullName',username, 5)
AS 'FullName',
REFERENCETABLE('user_data','Location',username, 5)
AS Location,
REFERENCETABLE('user_data','Manager',username, 5)
AS Manager
FROM events

Returns the full name (value), location (value), and manager (value) for the username (key) from user_data.

See more Reference data examples
REFERENCESETCONTAINS

Purpose
If a value is contained in a specific reference set, it returns true.

REFERENCESETCONTAINS
('Ref_Set', 'value', domainID)

Although the domainID is optional, in a domain-enabled environment, the search is limited to only shared reference data when the domainID is excluded.

Parameters
String, String, Integer

Example
SELECT
ASSETUSER(sourceip, NOW())
AS 'Source Asset User'
FROM flows
WHERE
REFERENCESETCONTAINS('Watchusers', username, 5)
GROUP BY "Source Asset User"
LAST 24 HOURS

Returns the asset user when the username (value) is included in the Watchusers reference set.

RULENAME

Purpose
Returns one or more rule names that are based on the rule ID or IDs that are passed in.

RULENAME(creeventlist)
RULENAME(3453)

Parameters
A single rule ID, or a list of rule IDs.

Example
SELECT * FROM events
WHERE RULENAME(creeEventList)
ILIKE '%my rule name%'

Returns events that trigger a specific rule name.

SELECT RULENAME(123)
FROM events

Returns rule name by the rule ID.

Time criteria in AQL queries

Define time intervals in your AQL queries by using START and STOP clauses, or use the LAST clause for relative time references.

Define the time settings that are 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 by using the following clauses and functions:

- "START" on page 41
- "STOP" on page 41
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 START in combination with STOP.

Examples

```sql
SELECT *
FROM events WHERE userName IS NULL
START '2014-04-25 15:51'
STOP '2014-04-25 17:00'
```


```sql
SELECT *
FROM events WHERE userName IS NULL
START '2014-04-25 15:51:20'
STOP '2014-04-25 17:00:20'
```

Returns results from: 2014-04-25 15:51:00 to 2014-04-25 17:00:59

```sql
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

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 STOP in combination with START.

Examples

```sql
SELECT * FROM events
WHERE username IS NULL
START '2016-04-25 14:00'
STOP '2016-04-25 16:00'
```
SELECT * FROM events
WHERE username IS NULL
START '2016-04-25 15:00:30'
STOP '2016-04-25 15:02:30'

Use any format 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.
Select * FROM events
START PARSEDATETIME('1 hour ago')
STOP PARSEDATETIME('now')
SELECT * FROM events
START PARSEDATETIME('1 day ago')
Select *
FROM events
WHERE logsourceid = '69'
START '2016-06-21 15:51:00'
STOP '2016-06-22 15:56:00'

LAST

You can pass a time interval to the LAST clause to specify a specific time to select data from.

The valid intervals are MINUTES, HOURS, and DAYS

Examples
SELECT * FROM events
LAST 15 MINUTES
SELECT * FROM events
LAST 2 DAYS
SELECT * FROM events
WHERE userName ILIKE '%dm%'
LIMIT 10
LAST 1 HOURS

Note: If you use a LIMIT clause in your query, you must place it before START and STOP clauses, for example,
SELECT *
FROM events
LIMIT 100
START '2016-06-28 10:00'
STOP '2016-06-28 11:00'

Time functions

Use the following time functions to specify the parse time for the query.

NOW

Purpose
Returns the current time that is expressed as milliseconds since the time 00:00:00 Coordinated Universal Time (UTC) on January 1, 1970.

Example
SELECT ASSETUSER(sourceip, NOW())
AS 'Asset user' FROM events

Find the user of the asset at this moment in time (NOW).
PARSEDATETIME

Purpose
Pass a time value to the parser, for example, PARSEDATETIME('time reference'). This 'time reference' is the parse time for the query.

Example
SELECT * FROM events
START PARSEDATETIME('1 hour ago')

AQL date and time formats

Use Ariel Query Language (AQL) date and time formats to represent times and dates in queries.

The following table lists the letters that represent date and time in AQL queries. This table is based on the SimpleDateFormat.

Table 6. Date and time formats

<table>
<thead>
<tr>
<th>Letter</th>
<th>Date or time parameter</th>
<th>Presentation</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Calendar year</td>
<td>Year</td>
<td>DATEFORMAT(starttime,'yy-MM-dd')</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Returns date format: 16-06-20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DATEFORMAT(starttime,'yyyy-MM-dd')</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Returns date format: 2016-06-20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SELECT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DATEFORMAT(devicetime,'yyyy-MM-dd')</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AS Log_Src_Date, QIDDESCRIPTION(qid)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AS 'Event Name'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FROM events</td>
</tr>
<tr>
<td>Y</td>
<td>Week year</td>
<td>Year</td>
<td>DATEFORMAT(starttime,'YY-MM-dd')</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The first and last days of a week year can have different calendar year values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Returns date format: 16-06-20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DATEFORMAT(starttime,'YYYY-MM-dd')</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Returns date format: 2016-06-20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SELECT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DATEFORMAT(starttime,'YYYY-MM-dd hh:mm')</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AS 'Start Time',</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DATEFORMAT(endtime,'YYYY-MM-dd hh:mm')</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AS End_time,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>QIDDESCRIPTION(qid)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AS 'Event Name'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FROM events</td>
</tr>
</tbody>
</table>

|        |                        |              | Returns start time, end time, and event name columns |
| M      | Month in year          | Month        | DATEFORMAT(starttime,'yyyy-MMM-dd') |
|        |                        |              | 3 or more letters are interpreted as text. |
|        |                        |              | Returns date format: 2016-June-20 |
|        |                        |              | 2 letters are interpreted as a number. |
|        |                        |              | Returns date format: 2016-Jun-20 |
|        |                        |              | DATEFORMAT(starttime,'yyyy-MM-dd') |
|        |                        |              | Returns date format: 2016-06-20 |
| w      | Week in year           | Number       | DATEFORMAT(starttime,'yyyy-ww-dd') |
|        |                        |              | Date example used is: 20-June-2016 |
|        |                        |              | Returns date format: 2016-26-20 |
|        |                        |              | Note: 26 is week 26 in year |
| W      | Week in month          | Number       | DATEFORMAT(starttime,'yyyy-WW-dd') |
|        |                        |              | Date example used is: 20-June-2016 |
|        |                        |              | Returns date format: 2016-04-20 |
|        |                        |              | Note: 04 is week 4 in month |
### Table 6. Date and time formats (continued)

<table>
<thead>
<tr>
<th>Letter</th>
<th>Date or time parameter</th>
<th>Presentation</th>
<th>Examples</th>
</tr>
</thead>
</table>
| D      | Day in year            | Number       | \(\text{DATEFORMAT(starttime,'yyyy-mm-DD')}\)  
\(\text{Returns date format: 2016-06-172} \)  
\(\text{Note: 172 is day number 172 in year}\) |
| d      | Day in month           | Number       | \(\text{DATEFORMAT(starttime,'yyyy-mm-dd')}\)  
\(\text{Returns date format: 2016-06-20}\) |
| F      | Day of week in month   | Number       | \(\text{DATEFORMAT(starttime,'yyyy-MM-FF')}\)  
\(\text{Returns date format: 2016-06-03} \)  
\(\text{Note: 03 is day 3 of week in month}\) |
| E      | Day name in week       | Text         | \(\text{DATEFORMAT(starttime,'yyyy-MM-EE')}\)  
\(\text{Returns date format: 2016-06-Mon}\) |
| a      | AM or PM               | Text         | \(\text{DATEFORMAT(starttime,'yyyy-MM-dd h a')}\)  
\(\text{2016-06-20 06 PM}\) |
| H      | Hour in day (0-23)     | Number       | \(\text{DATEFORMAT(starttime,'yyyy-MM-dd H')}\)  
\(\text{Returns date format: 2016-06-20 18} \)  
\(\text{Note: 18 is 18:00 hours}\) |
| k      | Hour in day (1-24)     | Number       | \(\text{DATEFORMAT(starttime,'yyyy-MM-dd k')}\)  
\(\text{Returns date format: 2016-06-20 18} \)  
\(\text{Note: 18 is 18:00 hours}\) |
| K      | Hour in AM/PM (0-11)   | Number       | \(\text{DATEFORMAT(starttime,'yyyy-MM-dd K a')}\)  
\(\text{Returns date format: 2016-06-20 6 PM} \)  
\(\text{Note: K = 6 and a = PM}\) |
| h      | Hour in AM/PM (1-12)   | Number       | \(\text{DATEFORMAT(starttime,'yyyy-MM-dd h a')}\)  
\(\text{Returns date format: 2016-06-20 6 PM} \)  
\(\text{Note: h = 6 and a = PM}\) |
| m      | Minute in hour         | Number       | \(\text{DATEFORMAT(starttime,'yyyy-MM-dd h:m a')}\)  
\(\text{Returns date format: 2016-06-20 6:10 PM} \)  
\(\text{Note: colon added in query to format time}\) |
| s      | Second in minute       | Number       | \(\text{DATEFORMAT(starttime,'yyyy-MM-dd h:m:s a')}\)  
\(\text{Returns date format: 2016-06-20 6:10:56 PM} \)  
\(\text{Note: colons added in query to format time}\) |
| S      | Millisecond            | Number       | \(\text{DATEFORMAT(starttime,'yyyy-MM-dd h:m:ss:s55 a')}\)  
\(\text{Returns date format: 2016-06-20 6:10:00:322 PM} \)  
\(\text{Note: colons added in query to format time}\) |
| z      | Time zone              | General Time zone | \(\text{DATEFORMAT(starttime,'yyyy-MM-dd h:m a z')}\)  
\(\text{Returns date format: 2016-06-20 6:10 PM GMT +1} \)  
\(\text{Note: colon added in query to format time}\) |
| Z      | Time zone              | RFC 822 time zone | \(\text{DATEFORMAT(starttime,'yyyy-MM-dd h:m a Z')}\)  
\(\text{Returns date format: 2016-06-20 6:10 PM GMT +1} \)  
\(\text{Note: colon added in query to format time}\) |
| X      | Time zone              | ISO 8601 time zone | \(\text{DATEFORMAT(starttime,'yyyy-MM-dd h:m a X')}\)  
\(\text{Returns date format: 2016-06-20 6:10 PM GMT +1} \)  
\(\text{Note: colon added in query to format time}\) |

### AQL subquery

Use an AQL subquery as a data source that is referred to, or searched by the main query. Use the FROM or IN clause to refine your AQL query by referring to the data that is retrieved by the subquery.
A subquery is a nested or inner query that is referenced by the main query. The subquery is available in the following formats:

- SELECT <field/s> FROM (<AQL query expression>)
  This query uses the FROM clause to search the output (cursor) of the subquery.
- SELECT <field/s> FROM events WHERE <field> IN (<AQL query expression>)
  This query uses the IN clause to specify the subquery results that match values from the subquery search. This subquery returns only one column. You can specify the results limit but the maximum is 10,000 results.

Subquery examples

The nested SELECT statement in parenthesis is the subquery. The subquery is run first and it provides the data that is used by the main query. The main query SELECT statement retrieves the user names from the output (cursor) of the subquery.

```
SELECT username FROM 
(SELECT * FROM events 
WHERE username IS NOT NULL 
LAST 60 MINUTES)
```

The following query returns records where the user name from the Ariel database matches values in the subquery.

```
SELECT * FROM events 
WHERE username IN 
(SELECT username FROM events 
LIMIT 10 LAST 5 MINUTES) LAST 24 HOURS
```

The following query returns records where the source IP address from the Ariel database matches the destination IP address in the subquery.

```
SELECT * FROM EVENTS 
WHERE sourceip IN 
(SELECT destinationip FROM events)
```

The following query returns records where the source IP address from the Ariel database matches the source IP addresses that are returned in the subquery. The subquery filters the data for the main select statement by locating internal hosts that interacted with high-risk entities. The query returns hosts that communicated with any hosts that interacted with high-risk entities.

```
SELECT sourceip AS 'Risky Hosts' FROM events 
WHERE destinationip IN (SELECT sourceip FROM events 
WHERE eventdirection = 'L2R' 
AND REFERENCESETCONTAINS('CriticalWatchList', destinationip) 
GROUP BY sourceip) 
GROUP BY sourceip last 24 hours
```

Grouping related events into sessions

Group events that are contextually related into sessions where you can observe event sequences and the outcomes of those event sequences. Gain insight into user activity and network activity by observing the sequence of events that occur in a session.

About this task

You can use events to tell you what a user did at a specific time, but you can use transactional sessions to tell you what the user did before and after an event. Transactions give you full detail such as a purchase on the internet, or an unauthorized login attempt.
The session ID is unique and is assigned to events in the same session. You define the session based on parameters such as time, user name, login, or any other criteria. You use the SESSION BY clause to create the unique sessions.

For example, use the transactional sessions to do these tasks:

- Define a user activity based on web-access events that includes a unique combination of activities.
- Group events by a specific user behavior session such as website visits, downloads, or emails sent.
- Record when users login to and logout of your network, and how long they log in for. The logout closes the related transaction that is initiated by the login.
- Pick an activity that you want to track and define the criteria for the session activity.

**Procedure**

1. To create sessions, use the SESSION BY clause by using the following format.

   `SESSION BY <TimeExpression> <AQL_expression_list> BEGIN <booleanExpression> END <booleanExpression>
   
   The following table describes the session parameters.

   **Table 7.**

<table>
<thead>
<tr>
<th>Session parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time &lt;TimeExpression&gt;</td>
<td>Time</td>
</tr>
<tr>
<td>&lt;AQL_expression_list&gt;</td>
<td>AQL expression list</td>
</tr>
<tr>
<td>BEGIN &lt;booleanExpression&gt;</td>
<td>Starts a new session</td>
</tr>
<tr>
<td>END &lt;booleanExpression&gt;</td>
<td>The END clause is optional, and is used to finish the session.</td>
</tr>
</tbody>
</table>
   
   The SessionId changes when any AQL expression value changes or when the BEGIN or END `booleanExpression` is TRUE.

2. To test an example, take the following steps:
   a. To go to the IBM Security QRadar API documentation page, from the Help menu, click Interactive API for Developers.
   b. Click 8.0 or the highest version to expand the menu.
   c. Click `/ariel` > `/searches`.
   d. Click the Post tab.
   e. Enter your AQL query in the Value field for the query_expression parameter. For example,
      ```
      Select sessionID, DATEFORMAT(starttime, 'YYYY-MM-dd HH:mm:ss')
      start_time, username, sourceip, category from events
      into <your_Cursor_Name> where username is not null
      SESSION BY starttime username, sourceip
      BEGIN category=16001
      start '2016-09-14 14:20' stop '2016-09-14 14:50'
      
      The <your_cursor_name> is any name that you want to use for the results output.
      ```
   f. Click Try it out.
   g. If the query runs without errors, the response code is 201.
   h. Click `/ariel` > `/searches` > `/[search_id]` > `/results`
   i. The 8.0 - GET - `/ariel/searches/[search_id]/results` page opens.
   j. In the Value field for the search_id parameter, type `<your_cursor_name>`.
   k. Select text/table for the Mime Type.
   l. Click Try it out.
Table 8. Query results

<table>
<thead>
<tr>
<th>sessionID</th>
<th>start_time</th>
<th>username</th>
<th>sourceip</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2016-09-14 14:42:03</td>
<td>admin</td>
<td>9.23.121.97</td>
<td>16003</td>
</tr>
<tr>
<td>1</td>
<td>2016-09-14 14:42:09</td>
<td>admin</td>
<td>9.23.121.97</td>
<td>16003</td>
</tr>
<tr>
<td>2</td>
<td>2016-09-14 14:42:10</td>
<td>admin</td>
<td>127.0.0.1</td>
<td>16003</td>
</tr>
<tr>
<td>2</td>
<td>2016-09-14 14:42:11</td>
<td>admin</td>
<td>127.0.0.1</td>
<td>16003</td>
</tr>
<tr>
<td>3</td>
<td>2016-09-14 14:42:27</td>
<td>joe_blogs</td>
<td>9.23.121.98</td>
<td>16001</td>
</tr>
<tr>
<td>4</td>
<td>2016-09-14 14:44:11</td>
<td>joe_blogs</td>
<td>9.23.121.98</td>
<td>16001</td>
</tr>
<tr>
<td>5</td>
<td>2016-09-14 14:44:35</td>
<td>root</td>
<td>127.0.0.1</td>
<td>4017</td>
</tr>
<tr>
<td>5</td>
<td>2016-09-14 14:44:35</td>
<td>root</td>
<td>127.0.0.1</td>
<td>3014</td>
</tr>
<tr>
<td>5</td>
<td>2016-09-14 14:44:55</td>
<td>root</td>
<td>127.0.0.1</td>
<td>4017</td>
</tr>
<tr>
<td>5</td>
<td>2016-09-14 14:44:55</td>
<td>root</td>
<td>127.0.0.1</td>
<td>3014</td>
</tr>
</tbody>
</table>

The categories represent specific activities in your event logs. A new session is started for every change of user name and source IP address values, for example, see sessionid 2 and sessionid 5. Also, a new session is created for category 16001, which occurs in sessionid 3 and sessionid 4.

Example

In this example events are returned and grouped by unique session ID, where the user joe_blogs logs in and starts a process between 4 PM and 11:30 PM on November 25.

```
select sessionId, DATEFORMAT(starttime, 'YYYY-MM-dd HH:mm:ss')
start_time, username, sourceip, category
from events
into <cursor_name>
where username='joe_blogs'
SESSION BY starttime username, sourceip
BEGIN category=16001
END category=16003
start '2016-11-25 16:00'
stop '2016-11-25 23:30'
```

A session is started when you get an event where the BEGIN expression is met OR the previous event ends the session.

A session is ended when you get an event where the END expression is true OR the next event starts a new session.

Event category 16001 indicates a user login or logout event on the Console, and event category 16003 indicates that a user initiated a process, such as starting a backup or generating a report. For a list of event categories, see the IBM Security QRadar Administration Guide.

Transactional query refinements

Refine transactional AQL queries by using the EXPLICIT expression with the BEGIN and END expressions. Also, use the TIMEOUT and TIMEWINDOW expressions to specify time intervals.

Use the EXPLICIT expression with the BEGIN and END expressions to apply more precise filtering to your transactional queries.

For example, you might use the BEGIN expression with the EXPLICIT END expression to capture several (BEGIN) unsuccessful login attempts, which are followed by an (EXPLICIT END) successful login.

Use the TIMEOUT and TIMEWINDOW expressions to apply time filters for the sessions in your transactional queries.
Expressions

Use the expressions that are described in the following to refine your transactional AQL query:

Table 9. AQL transactional query expressions

<table>
<thead>
<tr>
<th>Query expressions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td>A session is started when you get an event where the BEGIN expression is met or the previous event ends the session.</td>
</tr>
<tr>
<td>EXPLICIT BEGIN</td>
<td>Starts a new session only if the EXPLICIT BEGIN expression is true.</td>
</tr>
<tr>
<td>END</td>
<td>A session is ended when you get an event where the END expression is true or the next event starts a new session.</td>
</tr>
<tr>
<td>EXPLICIT END</td>
<td>Closes the current session only if the EXPLICIT END expression is true.</td>
</tr>
<tr>
<td>TIMEOUT</td>
<td>Closes the session when the specified TIMEOUT period elapses from the time that the previous event occurred to the time that the current event happened.</td>
</tr>
<tr>
<td>TIMEWINDOW</td>
<td>Tracks the session time. Closes the session when the specified TIMEWINDOW period elapses from the time that the first event occurred to the time that the current event happened.</td>
</tr>
</tbody>
</table>

Syntax

```
SESSION BY
<TimeExpression> <ExpressionList>
[EXPLICIT] BEGIN <booleanExpression>
[EXPLICIT] END <booleanExpression>
TIMEOUT <IntegerLiteral milliInSeconds>
TIMEWINDOW <IntegerLiteral SECONDS|MINUTES|HOURS|DAYS>
```

The following examples show the examples of results that you get by using different combinations of the available query expressions:

BEGIN and END expressions

A BEGIN expression starts a session when an event matches the BEGIN expression or the previous event ends the session.

An END expression ends a session when the END expression is true for an event or the next event starts a new session.

By using the EXPLICIT expression with the BEGIN and END expressions, you apply a more precise filter that refines the result set.

See the following examples of queries and results.

The following query example uses BEGIN and END expressions.

```
Select sessionId,
DATEFORMAT(starttime,'YYYY-MM-dd HH:mm:ss')
starttime, username, sourceip,
category from events into TR1
where username = 'user_x'
SESSION BY starttime username, sourceip
BEGIN category=16001
END category=16003
start '2016-12-10 16:00' stop '2016-12-10 23:30'
```

Event category 16001 indicates a user login or logout event on the Console, and event category 16003 indicates that a user initiated a process, such as starting a backup or generating a report.
The following table shows the results for the query that uses BEGIN and END.

Table 10. BEGIN and END query results

<table>
<thead>
<tr>
<th>sessionID</th>
<th>start_Time</th>
<th>user name</th>
<th>sourceip</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2016-12-10 16:00:06</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>1</td>
<td>2016-12-10 16:00:06</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>2</td>
<td>2016-12-10 16:00:06</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>3</td>
<td>2016-12-10 16:00:10</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>3</td>
<td>2016-12-10 16:00:11</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>3</td>
<td>2016-12-10 16:00:11</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>4</td>
<td>2016-12-10 16:00:14</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>5</td>
<td>2016-12-10 16:00:15</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>5</td>
<td>2016-12-10 16:00:15</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
</tbody>
</table>

**Note:** Session 2 consists of only one event that closes it (category 16003). A session that has one event is an exception and can happen.

**EXPLICIT BEGIN and END expressions**

Events are skipped when a session is not started and an event is not an EXPLICIT BEGIN event.

```
Select sessionID, 
DATEFORMAT(starttime,'YYYY-MM-dd HH:mm:ss') start_time, username, sourceip, 
category from events into TR2 
where username='user_x' 
SESSION BY starttime username, 
sourceip EXPLICIT BEGIN category=16001 
END category=16003 start '2016-12-10 16:00' 
stop '2016-12-10 23:30'
```

The following table shows the results for the query that uses EXPLICIT BEGIN and END.

Table 11. EXPLICIT BEGIN and END query results

<table>
<thead>
<tr>
<th>sessionID</th>
<th>start_Time</th>
<th>user name</th>
<th>sourceip</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2016-12-10 16:00:06</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>1</td>
<td>2016-12-10 16:00:06</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>2</td>
<td>2016-12-10 16:00:07</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>2</td>
<td>2016-12-10 16:00:07</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>3</td>
<td>2016-12-10 16:00:11</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>3</td>
<td>2016-12-10 16:00:11</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>4</td>
<td>2016-12-10 16:00:14</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>5</td>
<td>2016-12-10 16:00:15</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>5</td>
<td>2016-12-10 16:00:15</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
</tbody>
</table>

Only events that satisfy the EXPLICIT BEGIN expression are returned.

Session 2 and Session 4 in the EXPLICIT BEGIN and END don't satisfy the EXPLICIT BEGIN expression.
**BEGIN and EXPLICIT END**

Close current session only if the EXPLICIT END expression is true. There are no more checks for BEGIN events in the session when the EXPLICIT END expression is true.

Multiple BEGIN events in a single session can be associated with one EXPLICIT END expression. For example, you might use the EXPLICIT END expression for counting multiple failed login attempts that are followed by a successful login during a specific time interval (session timeout).

The following query example uses BEGIN and EXPLICIT END expressions.

```sql
Select sessionId,
DATEFORMAT(starttime,'YYYY-MM-dd HH:mm:ss')
starttime, username, sourceip,
category from events into TR3
where username = 'user_x'
SESSION BY starttime username, sourceip
BEGIN category=16001
EXPLICIT END category=16003
start '2016-12-10 16:00'
stop '2016-12-10 23:30'
```

The following table shows the results for the query that uses BEGIN and EXPLICIT END expressions.

**Table 12. BEGIN and EXPLICIT END query results**

<table>
<thead>
<tr>
<th>sessionID</th>
<th>start_Time</th>
<th>username</th>
<th>sourceip</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2016-12-10 16:00:06</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>1</td>
<td>2016-12-10 16:00:06</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>2</td>
<td>2016-12-10 16:00:07</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>2</td>
<td>2016-12-10 16:00:10</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>2</td>
<td>2016-12-10 16:00:10</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>3</td>
<td>2016-12-10 16:00:11</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>3</td>
<td>2016-12-10 16:00:11</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>4</td>
<td>2016-12-10 16:00:12</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>4</td>
<td>2016-12-10 16:00:12</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>4</td>
<td>2016-12-10 16:00:12</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>5</td>
<td>2016-12-10 16:00:13</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>4</td>
<td>2016-12-10 16:00:11</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
</tbody>
</table>

**EXPLICIT BEGIN and EXPLICIT END**

Events are ignored when a session is not started and an event is not an EXPLICIT BEGIN event.

Close current session only if the EXPLICIT END expression is true. There are no more checks for BEGIN events in the session when the EXPLICIT END expression is true.

The following query example uses both EXPLICIT BEGIN and EXPLICIT END expressions.

```sql
Select sessionId,
DATEFORMAT(starttime,'YYYY-MM-dd HH:mm:ss')
starttime, username, sourceip,
category from events into TR4
where username = 'user_x'
SESSION BY starttime username, sourceip
```

50    QRadar Ariel Query Language Guide
The following table shows the results for the query that uses both EXPLICIT BEGIN and EXPLICIT END expressions.

**Table 13. EXPLICIT BEGIN and EXPLICIT END query results**

<table>
<thead>
<tr>
<th>sessionID</th>
<th>start_T</th>
<th>user name</th>
<th>sourceip</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2016-12-10 16:00:06</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>1</td>
<td>2016-12-10 16:00:06</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>2</td>
<td>2016-12-10 16:00:10</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>2</td>
<td>2016-12-10 16:00:10</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>3</td>
<td>2016-12-10 16:00:11</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>3</td>
<td>2016-12-10 16:00:12</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>3</td>
<td>2016-12-10 16:00:12</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>4</td>
<td>2016-12-10 16:00:13</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>4</td>
<td>2016-12-10 16:00:14</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>5</td>
<td>2016-12-10 16:00:15</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>5</td>
<td>2016-12-10 16:00:15</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
</tbody>
</table>

**TIMEOUT**

Closes the session when the specified TIMEOUT period elapses from the time that the previous event occurred to the time that the current event happened. The current event becomes part of a new session. The TIMEOUT value is specified in milliseconds.

The following query example uses the TIMEOUT expression.

```sql
Select sessionid, DATEFORMAT(starttime,'YYYY-MM-dd HH:mm:ss.SSS') start_time, username, sourceip, category from events into TR5 where username='user_x' SESSION BY starttime username, sourceip BEGIN category=16001 EXPLICIT END category=16003 TIMEOUT 3600 start '2016-12-10 16:00' stop '2016-12-10 23:30'
```

The following table shows the results for the query that uses the TIMEOUT expression.

**Table 14. TIMEOUT query results**

<table>
<thead>
<tr>
<th>sessionID</th>
<th>start_T</th>
<th>user name</th>
<th>sourceip</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2016-12-10 16:00:06.716</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>2</td>
<td>2016-12-10 16:00:10.328</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
</tbody>
</table>

Sessionid 1 is ended and sessionid 2 is started because the TIMEOUT of 3600 is exceeded.
TIMEWINDOW

Tracks the session time. Closes the session when the specified TIMEWINDOW period elapses from the time that the first event occurred to the time that the current event happened. The current event becomes part of a new session. The TIMEWINDOW value can be specified in seconds, minutes, hours, or days.

The following query example uses the TIMEWINDOW expression.

```sql
Select sessionId,
DATEFORMAT(starttime,'YYYY-MM-dd HH:mm:ss.SSS')
startTime, username, sourceip,
category from events into TR6
where username='user_x'
SESSION BY starttime username, sourceip
BEGIN category=16001
EXPLICIT END category=16003
TIMEWINDOW 3000
start '2016-12-10 16:00'
stop '2016-12-10 23:30'
```

The following table shows the results for the query that uses the TIMEWINDOW expression.

<table>
<thead>
<tr>
<th>sessionID</th>
<th>start_Time</th>
<th>username</th>
<th>sourceip</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2016-12-10 16:00:06.415</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>1</td>
<td>2016-12-10 16:00:06.433</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>2</td>
<td>2016-12-10 16:00:06.716</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
<tr>
<td>3</td>
<td>2016-12-10 16:00:10.328</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16001</td>
</tr>
<tr>
<td>3</td>
<td>2016-12-10 16:00:06.328</td>
<td>user_x</td>
<td>10.2.2.10</td>
<td>16003</td>
</tr>
</tbody>
</table>

SessionID 1 is within the TIMEWINDOW expression time but sessionID 2 is ended because the TIMEWINDOW of 3600 is exceeded.

Conditional logic in AQL queries

Use conditional logic in AQL queries by using IF and CASE expressions.

Use conditional logic in your AQL queries to provide alternative options, depending on whether the clause condition evaluates to true or false.

CASE Statements

CASE expressions return a Boolean true or false result. When an expression is returned as true, the value of that CASE expression is returned and processing is stopped. If the Boolean result is false, then the value of the ELSE clause is returned.

In the following example, when the user name is root, the value of the CASE expression that is returned is Admin root. When the user name is admin, the value of the CASE expression that is returned is Admin user. If the CASE expressions return a Boolean false, the value of the ELSE clause is returned.

```sql
SELECT CASE username
WHEN 'root'
THEN 'Admin root'
WHEN 'admin'
THEN 'Admin user'
ELSE 'other' END FROM events
```

When the WHEN statement is true, the THEN statement is processed, otherwise processing finishes.
**IF, THEN, ELSE statements**

Statements between THEN and ELSE are processed when the IF statement is true.

In this example, when the IF condition is true, 'ADMIN' is returned when the user name is 'root', otherwise the user name is returned from the events log.

```sql
SELECT sourceip,
IF username = 'root'
THEN 'ADMIN'
ELSE username AS user
FROM events
```

In the following example, if the log has no user name, then get it from the asset model. Otherwise, the user name is returned from the events log.

```sql
SELECT sourceip,
IF username IS NULL
THEN ASSETUSER(sourceip)
ELSE username AS username
FROM events
GROUP BY username
LAST 2 DAYS
```

**Bitwise operators in AQL queries**

Enhance the filtering capability and performance of your AQL queries that include IP addresses by using bitwise operators. Specify filters at the IP address octet level to return specific results.

By filtering on octets in an IP address, you can refine the IP address search criteria.

For example, to search for specific device types whose last octet in a source IP address ends in 100, such as x.y.z.100, you can use the following query:

```sql
SELECT LONG(sourceip)AS long_ip,
sourceip
FROM events into <cursor_name>
WHERE (long_ip & 0xff000000)=0x0a000000
GROUP BY long_ip
ORDER BY long_ip
```

In the example, the `<sourceip>` is returned as an integer. The integer is used by the bitwise AND operator. The hexadecimal value `<ff>` in the last octet position for the source IP address specifies a filter in the corresponding IP address octet position of 0x000000<IP address octet hexidecimal value>. In this case, the hexadecimal value `<64>` is substituted for the decimal value 100 in the IP address.

The result is all source IP addresses that end in 100. The results can be a list for a specific device type for a company, if the last octet of all of the IP addresses is 100.

The following examples outline scenarios to use when you search with bitwise operators.

**Bitwise AND (&) examples**

Returns all IP addresses that match 10.xxx.xxx.xxx

```sql
SELECT LONG(sourceip)AS long_ip,
sourceip
FROM events into t1
WHERE (long_ip & 0xff000000)=0x0a000000
GROUP BY long_ip
LIMIT 50
```

Returns all IP addresses that match xxx.100.xxx.xxx
```
SELECT LONG(sourceip)AS long_ip,
      sourceip
FROM events into t2
WHERE (long_ip & 0x00ff0000)=0x0064000
GROUP BY long_ip
ORDER BY long_ip

Returns all IP addresses that match xxx.xxx.220.xxx
```

```
SELECT LONG(sourceip)AS long_ip,
      sourceip
FROM events into t3
WHERE (long_ip & 0x0000ff00)=0x000dc00
GROUP BY long_ip
ORDER BY long_ip

Returns all IP addresses that match xxx.xxx.xxx.1
```

```
SELECT LONG(sourceip)AS long_ip,
      sourceip
FROM events
WHERE (long_ip & 0x000000ff)=0x0000001
GROUP BY long_ip
ORDER BY long_ip

Bitwise NOT (~) examples
```

Use the following examples to convert each 1-bit value to a 0-bit value, or each 0-bit value to a 1-bit value, in a given binary pattern.

```
SELECT ~123456789
FROM events
LIMIT 1

Returns 123456790

SELECT ~0
FROM events
LIMIT 1

Returns -1

SELECT ~2147483647
FROM events
LIMIT 1

Returns -2147483648
```

```
Bitwise OR examples
```

Use the following examples compare two bits. If both bits have a value of "1", then the query returns a 1. If both bits have a value of "0", then the query returns a 0.

```
SELECT destinationip,
      LONG(destinationip),
      sourceip,
      LONG(sourceip)AS source_ip,
      LONG(destinationip)|source_ip
FROM events
WHERE destinationip='127.0.0.1'
LIMIT 1

SELECT destinationip,
      LONG(destinationip),
      sourceip,
      ~LONG(sourceip)AS not_source_ip,
      ~LONG(sourceip)|source_ip
FROM events
WHERE destinationip='127.0.0.1'
LIMIT 1
```
Bitwise XOR examples

The following examples can be used to take 2-bit patterns, or a pair of bits from each position, and convert them to either a 1 or a 0. If the bits are different, the result in that position is 1. If the bits are identical, the result in that position is 0.

SELECT 2147483647 # 2147483647 FROM events LIMIT 1

Returns 0

SELECT 12345 # 6789 AS A,
(~12345 & 6789) | (12345 & ~6789) AS B FROM events LIMIT 1

Returns 10940, 10940

ShiftLeft examples

The number of places to shift is given as the second argument to the shift operator.

SELECT -1 << 1 AS A FROM events LIMIT 1

Returns -2

SELECT 16 << 1 AS A FROM events LIMIT 1

Returns 128

ShiftRight examples

The operator >> uses the sign bit, which is the left-most bit, to fill the trailing positions after the shift. If the number is negative, then 1 is used as a filter and if the number is positive, then 0 is used as a filter.

SELECT 16 >> 3 AS A FROM events LIMIT 1

Returns 2

SELECT -32768 >> 15 AS A FROM events LIMIT 1

Returns 2
Returns -1

**ShiftRightUnsigned example**

Always fills 0 regardless of the sign of the number.

```sql
SELECT -1>>>33
FROM events
LIMIT 1
```

Returns 2147483647

Dividing by the power of 2.

```sql
SELECT (20+44)>>>1 A,
     (20+44)>>>2 B,
     (20+44)>>>3 C,
     (20+44)>>>4 D,
     (20+44)>>>5 E
FROM events
LIMIT 1
```

**CIDR IP addresses in AQL queries**

You can insert CIDR IP addresses (IPv4 or IPv6) in your AQL statements to query by IP address range, source IP, destination IP, or you can exclude specific CIDR IP addresses.

**Examples of CIDR IP addresses in AQL queries**

Query by source CIDR IP address, or by destination CIDR IP address.

```sql
SELECT * FROM flows
WHERE INCIDR('10.100.100.0/24',sourceip)
SELECT * FROM flows
WHERE INCIDR('10.100.100.0/24',destinationip)
SELECT * FROM flows
WHERE INCIDR('ff02:0:0:0:0:1:ff2f:29d6',destinationv6)
```

Query for flows that have a source or destination CIDR IP address of 10.100.100.0/24

```sql
SELECT * FROM flows
WHERE INCIDR('10.100.100.0/24',sourceip)
OR INCIDR('10.100.100.0/24',destinationip)
```

Query for events where 192.168.222.0/24 is not the source CIDR IP address.

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

Query for flows where 192.168.222.0/24 is not the destination CIDR IP address.

```sql
SELECT * FROM flows
WHERE NOT INCIDR('192.168.222.0/24',destinationip)
```

**Custom properties in AQL queries**

You can call a custom property directly in your AQL statements. If the custom property contains spaces you must use double quotation marks to encapsulate the custom property.

You must enable a custom property before you can use it in an AQL statement.
If the custom property is not enabled, you will be able to run your AQL query but you will not get results.

**Custom property example**

```
SELECT Bluecoat-cs-host, sourceip, Bluecoat-cs-uri  
FROM events  
WHERE LOGSOURCEGROUPNAME(devicegrouplist)  
ILIKE '%Proxies%'  
AND Bluecoat-cs-host ILIKE '%facebook.com%'  
GROUP BY sourceip
```

Bluecoat-cs-host is the host name from the client's URL that is requested.

Bluecoat-cs-uri is the original URL that is requested.

**System performance query examples**

You can use or edit examples of system performance AQL queries to run in your network.

Use the following query examples to get information about system performance in your network or edit these examples to build your own custom queries.

**Disk Utilization and CPU usage**

```
SELECT Hostname, "Metric ID", AVG(Value)  
AS Avg_Value, Element  
FROM events  
WHERE LOGSOURCENAME(logsourceid)  
ILIKE '%%health%%'  
AND "Metric ID"='SystemCPU'  
OR "Metric ID"='DiskUtilizationDevice'  
GROUP BY Hostname, "Metric ID", Element  
ORDER BY Hostname last 20 minutes
```

This query outputs the **Hostname**, **MetricID**, **Avg_Value**, and **Element** columns.

The **Avg_Value** column returns an average value for CPU usage and disk utilization.

**Disk Utilization by partition**

```
SELECT Hostname, AVG(Value) AS Disk_Usage, Element  
FROM events  
WHERE LOGSOURCENAME(logsourceid)  
ILIKE '%%health%%'  
AND "Metric ID"='DiskUsage'  
GROUP BY Hostname, Element  
ORDER BY Hostname  
LAST 2 HOURS
```

This query outputs the **Hostname**, **Disk_Usage**, and **Element** columns.

The **Disk_Usage** column returns a value for disk usage for the directories that are listed in the **Element** column.

**Disk usage in gigabytes (GB) per partition**

```
SELECT Element  
AS Partition_Name,  
MAX(value/(1024*1024*1024))  
AS 'Gigabytes_Used'  
FROM events
```

2 Ariel Query Language 57
WHERE "Metric ID"='DiskSpaceUsed'
GROUP BY element
ORDER BY Gigabytes_Used DESC
LAST 2 DAYS

This query outputs the Partition_Name and the Gigabytes_Used columns from the events database.

The Gigabytes_Used column returns a value for the gigabytes that are used by each partition that is listed in the Gigabytes_Used column for the last two days.

**Copying query examples from the AQL guide**

If you copy and paste a query example that contains single or double quotation marks from the AQL Guide, you must retype the quotation marks to be sure that the query parses.

**Events and flows query examples**

Use or edit query examples to create events and flows queries that you can use for your AQL searches.

Use the following query examples to get information about events and flows in your network or edit these examples to build your own custom queries.

**Event rates and flow rates for specific hosts**

SELECT AVG(Value), "Metric ID", Hostname
FROM events
WHERE LOGSOURCENAME(logsourceid) ILIKE '%%health%%'
AND ("Metric ID"='FlowRate' OR "Metric ID"='EventRate')
GROUP BY "Metric ID", Hostname
LAST 15 minutes

This query outputs the AVG_Value, Metric ID, and Hostname columns from the events or flows database for the last 15 minutes.

The AVG_Value column returns a value for the average flow or event rate over the last 15 minutes for the host that is named in the Hostname column.

**EPS rates by log source**

SELECT logsourcecname(logsourceid) AS 'MY Log Sources',
SUM(eventcount) / 2.0*60*60 AS EPS_Rates
FROM events
GROUP BY logsourceid
ORDER BY EPS_Rates DESC
LAST 2 HOURS

This query outputs My Log Sources, and EPS_Rates columns from events.

The My Log Sources column returns log source names and the EPS_Rates column returns the EPS rates for each log source in the last two hours.

**Event counts and event types per day**

SELECT
DATEFORMAT({ devicetime, 'dd-MM-yyyy'})
AS 'Date of log source',
QIDDESCRIPTION(qid)
AS 'Description of event', COUNT(*)

QRadar Ariel Query Language Guide
FROM events
WHERE devicetime > (now() - (7*24*3600*1000))
GROUP BY "Date of log source", qid
LAST 4 DAYS

This query outputs the Date of log source, Description of event, and count of event columns from events.

The date of the event, description of event, and count of events are returned for the last four days.

**Monitoring local to remote flow traffic by network**

```sql
SELECT sourceip,
LONG(SUM(sourcebytes+destinationbytes))
AS TotalBytes
FROM flows
WHERE flowdirection= 'L2R'
AND NETWORKNAME(sourceip)
ILIKE 'servers'
GROUP BY sourceip
ORDER BY TotalBytes
```

This query outputs the sourceip and TotalBytes columns.

The TotalBytes column returns the sum of the source and destination bytes that crosses from local to remote.

**Monitoring remote to local flow traffic by network**

```sql
SELECT sourceip,
LONG(SUM(sourcebytes+destinationbytes))
AS TotalBytes
FROM flows
WHERE flowdirection= 'R2L'
AND NETWORKNAME(sourceip)
ILIKE 'servers'
GROUP BY sourceip
ORDER BY TotalBytes
```

This query outputs the sourceip and TotalBytes columns.

The TotalBytes column returns the sum of the source and destination bytes from remote to local.

**Copying query examples from the AQL guide**

If you copy and paste a query example that contains single or double quotation marks from the AQL Guide, you must retype the quotation marks to be sure that the query parses.

**Reference data query examples**

Use AQL queries to get data from reference sets, reference maps, or reference tables. You can create and populate reference data by using rules to populate reference sets, by using external threat feeds, for example, LDAP Threat Intelligence App, or by using imported data files for your reference set.

Use the following examples to help you create queries to extract data from your reference data.

**Use reference tables to get external metadata for user names that show up in events**

```sql
SELECT
REFERENCETABLE('user_data','FullName',username) AS 'Full Name',
REFERENCETABLE('user_data','Location',username) AS 'Location',
```
Use the reference table to get external data such as the full name, location, and manager name for users who logged in to the network in the last 24 hours.

**Get the global user IDs for users in events who are flagged for suspicious activity**

```sql
SELECT
REFERENCEMAP('GlobalID_Mapping', username) AS 'Global ID',
REFERENCETABLE('user_data', 'FullName', 'Global ID') AS 'Full Name',
UNIQUECOUNT(username),
COUNT(*) AS 'Event count'
FROM events
WHERE RULENAME(creEventlist) ILIKE '%suspicious%'
GROUP BY "Global ID"
LAST 2 days
```

In this example, individual users have multiple accounts across the network. The organization requires a single view of a user’s activity. Use reference data to map local user IDs to a global ID. The query returns the user accounts that are used by a global ID for events that are flagged as suspicious.

**Use a reference map lookup to extract global user names for user names that are returned in events**

```sql
SELECT
QIDNAME(qid) as 'Event name',
starttime AS Time,
sourceip AS 'Source IP',
destinationip AS 'Destination IP',
username AS 'Event Username',
REFERENCEMAP('GlobalID_Mapping', username) AS 'Global User'
FROM events
WHERE "Global User" = 'John Ariel'
LAST 1 days
```

Use the reference map to look up the global user names for user names that are returned in events. Use the WHERE clause to return only events for the global user John Ariel. John Ariel might have a few different user names but these user names are mapped to a global user, for example, in an external identity mapping system, you can map a global user to several user names used by the same global user.

**Monitoring high network utilization by users**

```sql
SELECT
LONG(REFERENCETABLE('PeerGroupStats', 'average',
REFERENCEMAP('PeerGroup', username)))
AS PGave,
LONG(REFERENCETABLE('PeerGroupStats', 'stdev',
REFERENCEMAP('PeerGroup', username)))
AS PGstd,
SUM(sourcebytes+destinationbytes) AS UserTotal
FROM flows
WHERE flowtype = 'L2R'
GROUP BY UserTotal
HAVING UserTotal > (PGave + 3*PGstd)
```

Returns user names where the flow utilization is three times greater than the average user.
You need a reference set to store network utilization of peers by user name and total bytes.

**Threat ratings and categories**

```sql
SELECT
  REFERENCETABLE('ip_threat_data', 'Category', 'destinationip')
AS 'Threat Category',
  REFERENCETABLE('ip_threat_data', 'Rating', 'destinationip')
AS 'Threat Rating',
  UNIQUECOUNT(sourceip) AS 'Source IP Count',
  UNIQUECOUNT(destinationip) AS 'Destination IP Count'
FROM events
GROUP BY "Threat Category", "Threat Rating" LAST 24 HOURS
```

Returns the threat category and the threat rating.

You can look up reference table threat data and include it in your searches.

**Copying query examples from the AQL guide**

If you copy and paste a query example that contains single or double quotation marks from the AQL Guide, you must retype the quotation marks to be sure that the query parses.

**User and network monitoring query examples**

Use query examples to help you create your user and network monitoring query AQL queries.

Use the following examples to monitor your users and network, or you can edit the queries to suit your requirements.

**Find users who used the VPN to access the network from three or more IP addresses in a 24-hour period**

```sql
SELECT
  username,
  UNIQUECOUNT(sourceip) AS 'Source IP count'
FROM events
WHERE LOGSOURCENAME(logsourceid) ILIKE '%VPN%'
AND username IS NOT NULL
GROUP BY username
HAVING "Source IP count" >= 3
ORDER BY "Source IP count" DESC
LAST 24 HOURS
```

This query outputs the `username` and `Source IP count` columns.

The `username` column returns the names of users who used the VPN to access the network from three or more IP addresses in the last 24 hours.

**Find users who used the VPN from more that one geographic location in 24 hours**

```sql
SELECT
  username, UNIQUECOUNT(geographiclocation)
AS 'Count of locations'
FROM events
WHERE LOGSOURCENAME(logsourceid) ILIKE '%VPN%'
AND geographiclocation <> 'other location'
AND username IS NOT NULL
```

2 Ariel Query Language 61
GROUP BY username
HAVING "Count of locations" > 1
ORDER BY "Count of locations"
DESC
LAST 3 DAYS

This query outputs the username and Count of locations columns.

The username column returns the names of users who used the VPN from more than one location that is not called 'other location' in the last 24 hours.

Monitoring local to remote flow traffic by network

SELECT sourceip,
LONG(SUM(sourcebytes+destinationbytes))
AS TotalBytes
FROM flows
WHERE flowdirection= 'L2R'
AND NETWORKNAME(sourceip)
ILIKE 'servers'
GROUP BY sourceip
ORDER BY TotalBytes

This query outputs the sourceip and TotalBytes columns.

The TotalBytes column returns the sum of the source and destination bytes that crosses from local to remote.

Monitoring remote to local flow traffic by network

SELECT sourceip,
LONG(SUM(sourcebytes+destinationbytes))
AS TotalBytes
FROM flows
WHERE flowdirection= 'R2L'
AND NETWORKNAME(sourceip)
ILIKE 'servers'
GROUP BY sourceip
ORDER BY TotalBytes

This query outputs the sourceip and TotalBytes columns.

The TotalBytes column returns the sum of the source and destination bytes from remote to local.

Application usage by application name, users, and flows traffic

SELECT sourceip
AS Source_IP,
FIRST(destinationip)
AS Destination_IP,
APPLICATIONNAME(applicationid)
AS Application,
DATEFORMAT(lastpackettime, 'dd-MM-yyyy hh:mm:ss')
AS 'Start Time',
FIRST(sourcebytes)
AS Source_Bytes,
ASSETUSER(sourceip, NOW()) AS Src_Asset_User
FROM flows
GROUP BY Source_IP
ORDER BY Source_Bytes DESC

This query outputs data about your asset users, application names, and flow data. Use this query to report specific user activity or application usage, or to build a variation of this query to achieve your desired results.
Location of assets

```
SELECT ASSETPROPERTY('Location', sourceip) 
AS asset_location, 
COUNT(*) 
FROM events 
GROUP BY asset_location 
LAST 1 days
```

This query outputs the asset_location and count columns.

The asset location column returns the location of the assets.

Copying query examples from the AQL guide

If you copy and paste a query example that contains single or double quotation marks from the AQL Guide, you must retype the quotation marks to be sure that the query parses.

Event, flow, and simarc fields for AQL queries

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

Supported event fields for AQL queries

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

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adekey</td>
<td>Ade key</td>
</tr>
<tr>
<td>adevalue</td>
<td>Ade value</td>
</tr>
<tr>
<td>category</td>
<td>Low-level category</td>
</tr>
<tr>
<td>creEventList</td>
<td>Matched custom rule</td>
</tr>
<tr>
<td>credibility</td>
<td>Credibility</td>
</tr>
<tr>
<td>destinationMAC</td>
<td>Destination MAC</td>
</tr>
<tr>
<td>destinationPort</td>
<td>Destination port</td>
</tr>
<tr>
<td>destinationv6</td>
<td>IPv6 destination</td>
</tr>
<tr>
<td>destinationaddress</td>
<td>Destination address</td>
</tr>
<tr>
<td>destinationip</td>
<td>Destination IP</td>
</tr>
<tr>
<td>sourceaddress</td>
<td>Source address</td>
</tr>
<tr>
<td>deviceTime</td>
<td>Log source time</td>
</tr>
<tr>
<td>deviceType</td>
<td>Log source type</td>
</tr>
<tr>
<td>devicegrouplist</td>
<td>Device group list</td>
</tr>
<tr>
<td>domainID</td>
<td>Domain ID</td>
</tr>
<tr>
<td>duration</td>
<td>Duration</td>
</tr>
<tr>
<td>endTime</td>
<td>End time</td>
</tr>
<tr>
<td>eventCount</td>
<td>Event count</td>
</tr>
<tr>
<td>Field name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>eventDirection</td>
<td>Event direction:</td>
</tr>
<tr>
<td></td>
<td><em>local-to-Local (L2L)</em></td>
</tr>
<tr>
<td></td>
<td><em>local-to-remote (L2R)</em></td>
</tr>
<tr>
<td></td>
<td><em>remote-to-local (R2L)</em></td>
</tr>
<tr>
<td></td>
<td><em>remote-to-remote (R2R)</em></td>
</tr>
<tr>
<td>geographiclocation</td>
<td>Geographic location</td>
</tr>
<tr>
<td>sourcegeographiclocation</td>
<td>Source geographic location</td>
</tr>
<tr>
<td>destinationgeographiclocation</td>
<td>Destination geographic location</td>
</tr>
<tr>
<td>hasIdentity</td>
<td>Has identity</td>
</tr>
<tr>
<td>hasOffense</td>
<td>Associated with offense</td>
</tr>
<tr>
<td>highLevelCategory</td>
<td>High-level category</td>
</tr>
<tr>
<td>identityhostname</td>
<td>Identity host name</td>
</tr>
<tr>
<td>identityip</td>
<td>Identity IP address</td>
</tr>
<tr>
<td>isduplicate</td>
<td>Is duplicate</td>
</tr>
<tr>
<td>isCREEEvent</td>
<td>Is custom rule event</td>
</tr>
<tr>
<td>logsourceid</td>
<td>Log source ID</td>
</tr>
<tr>
<td>magnitude</td>
<td>Magnitude</td>
</tr>
<tr>
<td>pcappacket</td>
<td>PCAP packet</td>
</tr>
<tr>
<td>partialMatchList</td>
<td>Partial match list</td>
</tr>
<tr>
<td>payload</td>
<td>Payload</td>
</tr>
<tr>
<td>postNatDestinationIP</td>
<td>Destination IP after NAT</td>
</tr>
<tr>
<td>postNatDestinationPort</td>
<td>Destination port after NAT</td>
</tr>
<tr>
<td>postNatSourceIP</td>
<td>Source IP after NAT</td>
</tr>
<tr>
<td>postNatSourcePort</td>
<td>Source port after NAT</td>
</tr>
<tr>
<td>preNatDestinationIP</td>
<td>Destination IP before NAT</td>
</tr>
<tr>
<td>preNatDestinationPort</td>
<td>Destination port before NAT</td>
</tr>
<tr>
<td>preNatSourceIP</td>
<td>Source IP before NAT</td>
</tr>
<tr>
<td>preNatSourcePort</td>
<td>Source port before NAT</td>
</tr>
<tr>
<td>protocolid</td>
<td>Protocol</td>
</tr>
<tr>
<td>processorId</td>
<td>Event Processor ID</td>
</tr>
<tr>
<td>qid</td>
<td>Event name ID</td>
</tr>
<tr>
<td>relevance</td>
<td>Relevance</td>
</tr>
<tr>
<td>severity</td>
<td>Severity</td>
</tr>
<tr>
<td>sourceIP</td>
<td>Source IP</td>
</tr>
<tr>
<td>sourceMAC</td>
<td>Source MAC</td>
</tr>
<tr>
<td>sourcePort</td>
<td>Source port</td>
</tr>
<tr>
<td>sourcev6</td>
<td>IPv6 source</td>
</tr>
<tr>
<td>startTime</td>
<td>Start time</td>
</tr>
<tr>
<td>isunparsed</td>
<td>Event is unparsed</td>
</tr>
</tbody>
</table>
Table 16. Supported event fields for AQL queries (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userName</td>
<td>User name</td>
</tr>
</tbody>
</table>

Supported flow fields for AQL queries

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

Table 17. Supported flow fields for AQL queries

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationId</td>
<td>Application ID</td>
</tr>
<tr>
<td>category</td>
<td>Category</td>
</tr>
<tr>
<td>credibility</td>
<td>Credibility</td>
</tr>
<tr>
<td>destinationASN</td>
<td>Destination ASN</td>
</tr>
<tr>
<td>destinationBytes</td>
<td>Destination bytes</td>
</tr>
<tr>
<td>destinationDSCP</td>
<td>Destination DSCP</td>
</tr>
<tr>
<td>destinationFlags</td>
<td>Destination flags</td>
</tr>
<tr>
<td>destinationIP</td>
<td>Destination IP</td>
</tr>
<tr>
<td>destinationIfIndex</td>
<td>Destination if index</td>
</tr>
<tr>
<td>destinationPackets</td>
<td>Destination packets</td>
</tr>
<tr>
<td>destinationPayload</td>
<td>Destination payload</td>
</tr>
<tr>
<td>destinationPort</td>
<td>Destination port</td>
</tr>
<tr>
<td>destinationPrecedence</td>
<td>Destination precedence</td>
</tr>
<tr>
<td>destinationv6</td>
<td>IPv6 destination</td>
</tr>
<tr>
<td>domainID</td>
<td>Domain ID</td>
</tr>
<tr>
<td>fullMatchList</td>
<td>Full match list</td>
</tr>
<tr>
<td>firstPacketTime</td>
<td>First packet time</td>
</tr>
<tr>
<td>flowBias</td>
<td>Flow bias</td>
</tr>
<tr>
<td>flowDirection</td>
<td>Flow direction</td>
</tr>
<tr>
<td></td>
<td>local-to-local (L2L)</td>
</tr>
<tr>
<td></td>
<td>local-to-remote (L2R)</td>
</tr>
<tr>
<td></td>
<td>remote-to-local (R2L)</td>
</tr>
<tr>
<td></td>
<td>remote-to-remote (R2R)</td>
</tr>
<tr>
<td>flowInterfaceID</td>
<td>Flow interface ID</td>
</tr>
<tr>
<td>flowSource</td>
<td>Flow Source</td>
</tr>
<tr>
<td>flowType</td>
<td>Flow type</td>
</tr>
<tr>
<td>geographic</td>
<td>Matches geographic location</td>
</tr>
<tr>
<td>hasDestinationPayload</td>
<td>Has destination payload</td>
</tr>
<tr>
<td>hasOffense</td>
<td>Has offense payload</td>
</tr>
<tr>
<td>hasSourcePayload</td>
<td>Has source payload</td>
</tr>
<tr>
<td>icmpCode</td>
<td>Icmp code</td>
</tr>
<tr>
<td>icmpType</td>
<td>ICMP type or code</td>
</tr>
</tbody>
</table>
### Table 17. Supported flow fields for AQL queries (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flowInterface</td>
<td>Flow interface</td>
</tr>
<tr>
<td>intervalId</td>
<td>Interval ID</td>
</tr>
<tr>
<td>isDuplicate</td>
<td>Duplicate event</td>
</tr>
<tr>
<td>lastPacketTime</td>
<td>Last packet time</td>
</tr>
<tr>
<td>partialMatchList</td>
<td>Partial match list</td>
</tr>
<tr>
<td>protocolId</td>
<td>Protocol ID</td>
</tr>
<tr>
<td>qid</td>
<td>Qid</td>
</tr>
<tr>
<td>processorID</td>
<td>Event processor ID</td>
</tr>
<tr>
<td>relevance</td>
<td>Relevance</td>
</tr>
<tr>
<td>retentionBucket</td>
<td>Retention bucket dummy</td>
</tr>
<tr>
<td>severity</td>
<td>Severity</td>
</tr>
<tr>
<td>sourceASN</td>
<td>Source ASN</td>
</tr>
<tr>
<td>sourceBytes</td>
<td>Source bytes</td>
</tr>
<tr>
<td>sourceDSCP</td>
<td>Source DSCP</td>
</tr>
<tr>
<td>sourceFlags</td>
<td>Source flags</td>
</tr>
<tr>
<td>sourceIP</td>
<td>Source IP</td>
</tr>
<tr>
<td>sourceIfIndex</td>
<td>Source if index</td>
</tr>
<tr>
<td>sourcePackets</td>
<td>Source packets</td>
</tr>
<tr>
<td>sourcePayload</td>
<td>Source payload</td>
</tr>
<tr>
<td>sourcePort</td>
<td>Source port</td>
</tr>
<tr>
<td>sourcePrecedence</td>
<td>Source precedence</td>
</tr>
<tr>
<td>sourcev6</td>
<td>IPv6 source</td>
</tr>
<tr>
<td>startTime</td>
<td>Start time</td>
</tr>
<tr>
<td>viewObjectPair</td>
<td>View object pair</td>
</tr>
</tbody>
</table>

### Supported simarc fields for AQL queries

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

### Table 18. Supported simarc fields for AQL queries

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destinationPort</td>
<td>Destination port key creator</td>
</tr>
<tr>
<td>destinationType</td>
<td>Destination type key creator</td>
</tr>
<tr>
<td>deviceId</td>
<td>Device key creator</td>
</tr>
<tr>
<td>direction</td>
<td>Direction key creator</td>
</tr>
<tr>
<td>eventCount</td>
<td>Event count key creator</td>
</tr>
<tr>
<td>eventFlag</td>
<td>Flag key creator</td>
</tr>
<tr>
<td>applicationId</td>
<td>Application ID key creator</td>
</tr>
<tr>
<td>flowCount</td>
<td>Flow count key creator</td>
</tr>
<tr>
<td>destinationBytes</td>
<td>Destination bytes key creator</td>
</tr>
<tr>
<td>flowSource</td>
<td>Flow source key creator</td>
</tr>
<tr>
<td>Field name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>sourceBytes</td>
<td>Source bytes key creator</td>
</tr>
<tr>
<td>lastPacketTime</td>
<td>Time key creator</td>
</tr>
<tr>
<td>protocolId</td>
<td>Protocol key creator</td>
</tr>
<tr>
<td>source</td>
<td>Source key creator</td>
</tr>
<tr>
<td>sourceType</td>
<td>Source type key creator</td>
</tr>
<tr>
<td>sourceRemoteNetwork</td>
<td>Source remote network key creator</td>
</tr>
<tr>
<td>destinationRemoteNetwork</td>
<td>Destination remote network key creator</td>
</tr>
<tr>
<td>sourceCountry</td>
<td>Source geographic key creator</td>
</tr>
<tr>
<td>destinationCountry</td>
<td>Destination geographic key creator</td>
</tr>
<tr>
<td>destination</td>
<td>Destination key creator</td>
</tr>
</tbody>
</table>
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

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 jurisdictions 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:
IBM Director of Licensing  
IBM Corporation  
North Castle Drive, MD-NC119  
Armonk, NY 10504-1785  
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM’s suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

**Trademarks**

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information” at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

![Java Compatible](https://www.ibm.com/soa/images/errors/javalogo.png)

**Terms and conditions for product documentation**

Permissions for the use of these publications are granted subject to the following terms and conditions.
Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

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](http://www.ibm.com/privacy) and IBM’s Online Privacy Statement at [http://www.ibm.com/privacy/details](http://www.ibm.com/privacy/details) the section entitled “Cookies, Web Beacons and Other

---

**General Data Protection Regulation**

Clients are responsible for ensuring their own compliance with various laws and regulations, including the European Union General Data Protection Regulation. Clients are solely responsible for obtaining advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulations that may affect the clients’ business and any actions the clients may need to take to comply with such laws and regulations. The products, services, and other capabilities described herein are not suitable for all client situations and may have restricted availability. IBM does not provide legal, accounting or auditing advice or represent or warrant that its services or products will ensure that clients are in compliance with any law or regulation.

Learn more about the IBM GDPR readiness journey and our GDPR capabilities and Offerings here: https://ibm.com/gdpr
Index

A
AQL  5
Ariel Query Language  5

W
WHERE clause  11

C
count information  v
COUNT function  16
customer support  v

D
description  v
documentation  v

E
events and flows  63

F
field list  63
functions
  Date and time format  43
  fields  5

G
GROUP BY  12

H
HAVING  14

L
LIKE clause  15

N
network administrator  v

O
ORDER BY clause  15

S
SELECT clause  11
Start and Stop clauses  40

T
technical library  v