Oracle® Cloud
Using Oracle Log Analytics Search

E61253-36
June 2022

Oracle Cloud Using Oracle Log Analytics Search,

E61253-36

Primary Author: Oracle Corporation

Contents

Preface

Audience v

Documentation Accessibility v

Related Resources v

Conventions v

1 About Oracle Log Analytics Search

2 Explore Log Data

Use the Oracle Log Analytics User Interface 2-2

Formulate Queries Using the Oracle Log Analytics UI 2-3

Write Search Queries 2-4

- Specify Entities 2-4
- Use Keywords, Phrases, and Wildcards 2-5
- Use Comparison Operators 2-6
- Use Boolean Expressions 2-7
- Understand the Search Commands 2-8
- Write Sub-Queries 2-11

A Command Reference

Addfields Command A-2

Addinsights Command A-3

Bottom Command A-4

Bucket Command A-5

Classify Command A-6

Cluster Command A-7

Clustercompare Command A-7

Clusterdetails Command A-9

Clustersplit Command A-9

Compare Command A-10

Createview Command A-12
Distinct Command A-12
Eval Command A-13
Eventstats Command A-23
Fields Command A-28
Fieldsummary Command A-29
Head Command A-30
Highlightgroups Command A-30
Highlightrows Command A-31
Highlight Command A-32
Link Command A-33
Lookup Command A-34
Map Command A-35
NLP Command A-36
Regex Command A-37
Rename Command A-39
Search Command A-40
SearchLookup Command A-42
Sort Command A-42
Stats Command A-43
Tail Command A-46
Timecluster Command A-46
Timestats Command A-47
Top Command A-49
Where Command A-49

B Log Queries: Quick Reference

Preface

Using Oracle Log Analytics Search describes how to create and edit queries in the Oracle Log Analytics user interface, to filter through all available log data and return only that data which you wish to view.

- Audience
- Documentation Accessibility
- Related Resources
- Conventions

Audience

Using Oracle Log Analytics Search is intended for Oracle Log Analytics users who want to search and monitor log data across the enterprise.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Resources

For more information, see these Oracle resources:

- About Oracle Log Analytics

Conventions

This table describes the text conventions used in this document.

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.

Convention	Meaning
italic	Italic type indicates the book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph. URLs, code in examples, text that appears on the screen, or text that you enter.

1

About Oracle Log Analytics Search

Oracle Log Analytics enables you to filter through and analyze vast amounts of log data across your enterprise databases, through a single, unified, and customizable display that's easy to read and navigate. Use the in-built Search feature to filter through all your available log data and return specific log entries.

Oracle Log Analytics Search helps you drill down to specific log entries, enabling focussed analysis and monitoring across your enterprise. Use the Oracle Log Analytics query language to formulate your Search queries, which will retrieve log entries specific to the problem you are troubleshooting.

The Oracle Log Analytics query language enables you to:

- Filter and explore all available log data
- Perform root cause analysis
- Run statistical analysis on selected entities
- Generate reports
- Save search queries for later use
- Retrieve saved searches to build dashboards

You can construct your Search query by either dragging elements from the **Field** panel and dropping them in the appropriate sections in the **Visualize** column, or by directly entering your query in the **Search** field. For more information about using the user interface to formulate your queries, see Example Scenario: Detect Anomalies Using Outliers in *Using Oracle Log Analytics*.

2

Explore Log Data

Oracle Log Analytics enables you to filter and analyze all available log data. By formulating and running a query, you can search all available log data and ensure you view only that specific data you are looking for.

The Oracle Log Analytics Search query is made up of multiple elements. Consider the following query:

```
'Target Type' IN ('Database Instance','Automatic Storage
Management','Listener','Cluster') AND Severity IN ('ERROR','SEVERE') |stats
count as 'Error Count' by target |top limit=10 'Error Count'
```

This query has the following elements:

- **Entities:** Entities include host machines, databases, and other components that can be managed and monitored by Oracle Log Analytics.
- **Commands:** This is a specific action to be performed. The first and implicit command in a query is the `search` command.
- **Keywords and Phrases:** A keyword is a single word, while a phrase contains more than one word, usually enclosed in quotation marks. These specify what exact words to look for. For example, specifying the phrases `‘Database Instance’`, and `‘Automatic Storage Management’` in the query ensure that logs not containing the specified phrases are filtered out.
- **Conditions:** These are specific criteria the data must meet, to be returned as a result after the query is run. For instance, in the example query, `AND` is a Boolean expression, where the results will return those logs which fulfil the conditions specified for `Target Type` and `Severity`.
- **Functions:** Functions specify a task that needs to be completed on the data. For example, in the command, `count` is a function which counts the rows in the results returned from running the first part of the query.

Using the Oracle Log Analytics search language, you can apply a second level filter to the data through the use of the pipe command (`|`). The command to the left of the first pipe character is the first level filter, while the command to the right of the pipe character is the second level filter. The results returned after running the first command are further refined according to the command to the right of the pipe command. You can use as many pipe characters as necessary to retrieve only that information which is necessary.

For example, consider the following query:

```
‘log source’ in (‘FMW WLS Server Access Log’,’FMW OHS Access Log’)
|stats count(URI) as ‘Request Count’ by URI | top limit=10 ‘Request Count’
```

When this query is run, Oracle Log Analytics performs the following actions:

1. Identifies data with `log source` within the logs `‘FMW WLS Server Access Log’` and `’FMW OHS Access Log’`.
2. Returns the count of rows from the results of step 1, where the value of the field `Request Count` for each URI is not null.
3. Lists the top 10 URIs with the highest `Request Count`.

It is good practice to begin a compound query with a general command, and make the query more specific to the right side of each pipe character.

Topics

- Use the Oracle Log Analytics User Interface
- Formulate Queries Using the Oracle Log Analytics UI
- Write Search Queries

Use the Oracle Log Analytics User Interface

The Oracle Log Analytics user interface enables you to formulate your Search query.

You can use the following elements of the UI to formulate your Search query:

- **Search bar:** Your search query is displayed here. You can directly edit the text in this field to further refine your search results.

  The search bar grows or shrinks based on the number of lines added to the query. It can have a maximum of 21 lines and a minimum of 1 line. Some of the custom shortcuts available are:

  – `Ctrl + i`: Indent every line of text present in the editor. Note that the uppercase `I` opens the debugger.

  – `Ctrl + Enter`: Execute the query displayed in the editor

  – `Ctrl + Space`: Display the auto-complete list of options based on the cursor position

  – `Ctrl + Z`: Undo the last edit

  – `Ctrl + Y`: Redo the last edit

  – `Ctrl + D`: Delete current line

  Note: **Do not** use the `SHIFT` key unless specified.

  Position the cursor in the open or closing brackets to view the matching element highlighted. The elements that can be highlighted are `( )` and `[ ]`.

  The search bar supports two different themes, color and gray-scale. You can change the themes dynamically by changing the option in the help pop-up.

- **Field:** The **Field** panel is divided into the following sections:

  – The **Pinned** attributes let you filter log data based on:

    * Log sources, such as database logs, Oracle WebLogic Server logs, and so on.

    * Log entities, which are the actual log file names.

        * Labels, which are tags added to log entries when log entries match specific defined conditions.
        * Upload names of log data uploaded on demand.

      By default, the entities and collection details are available in the Pinned bucket of the **Fields** panel for filtering. You can pin additional fields to the Pinned bucket depending on your usage. Once pinned, the fields are moved to the Pinned bucket. You can unpin any field and remove it from the Pinned bucket and move it back to the Interesting or Other bucket.

    – Based on your search and queries, Oracle Log Analytics automatically adds fields to the **Interesting** bucket for your quick reference. You can pin a field that's available under Interesting bucket. The pinned field then gets moved to the Pinned bucket.
    – You can pin any field in the **Other** bucket and move it to the Pinned bucket. If you use a field from the Other bucket in your search or query, then it's moved to the Interesting bucket.

      The selected options are automatically added to the query in the **Search** bar.

- **Visualize:** In this pane, you can select how you would prefer to view the search results. In the **Group by** field, you can decide what metrics to group the results by.
- **Save:** Use this button to save the search query that is currently in the **Search** field, to be run at a later time.
- **Open:** Use this button to view previously saved search queries. You can run these queries and get current results, or you can use these queries to create dashboards.
- **New:** Use this button to start a new search query.
- **Export:** Use this button to export the result of the current search query in a file of the `Comma-seperated Values`(CSV) or `JavaScript Object Notation`(JSON) format.
- **Run:** Use this button to run the query which is currently in the **Search** field.
- **Time Selector:** Use the Time Selector to specify the time period.
- **Visualization Pane:** The results of the search query are displayed in this pane. The filtered information in this pane loads when the query in the **Search** field is run. By clicking on any area in the chart in the visualization pane, you can drill down into the search query and update it.

Formulate Queries Using the Oracle Log Analytics UI

You can use the Oracle Log Analytics user interface to formulate your Search query.

By default, the query `* | stats count by 'log source'` is specified in the **Search** field.

To view data for specific **Entities**, complete the following steps:

1. Under the heading **Entities** in the Fields panel, select **Entity** or **Entity Type**, depending on how you wish to view the entities. This groups the registered databases on the basis of the selection you have made. For example, if you select **Entity Type**, then the selected entities are grouped according to their types.
2. Select the **Entity** or **Entity Type** for which you wish to view the data.
3. Click **Submit**.

   The button **clear** appears next to the entity you have selected, and the data displays in the visualization pane.

To view data for a specific **Field**, complete the following steps:

1. Select the type of field from the **Fields** panel, and under **Pinned** attributes, **Interesting** bucket, or **Other** bucket.
2. Select the **Label**, **Log Entity**, **Log Source**, **Owner**, or **Upload Name** for which you wish to view the data. You can select more than one **Label**, **Log Entity**, **Log Source**, **Owner**, or **Upload Name**.
3. Click **Submit**.

   Data for the field is loaded in the visualization pane.

Write Search Queries

You can specify entities, keywords, phrases or wildcards, comparison operators, Boolean expressions, functions, and time to create your Oracle Log Analytics search query.

To use the Search feature in Oracle Log Analytics, you must formulate a search query and enter it in the **Search** field.

Topics:

- Specify Entities
- Use Keywords, Phrases, and Wildcards
- Use Comparison Operators
- Use Boolean Expressions
- Understand the Search Commands
- Write Sub-Queries

Specify Entities

If you know for which entity you want to view information, then you can specify the entity in your query string.

Queries can optionally be bounded by some set of entities, for example,

- Return all fatal logs discovered in entities that are members of *MyProductionDatabases* group
- Return the count of logs for entities that are used by *MyDatabase* target

Here are some examples of specifying the entities using the JSON:

Return logs for entities *entity_abc* and *entity_xyz*:

```
"targetFilter":   {
 "filters":[{
   "name":"meId",
   "operator":"isEqual",
   "values":["entity_abc","entity_xyz"]
 }]
}
```

Return logs for entities that are members of *entity_abc*:

```
"targetFilter":   {
 "filters":[{
   "name":"memberOf",
   "operator":"isEqual",
   "values":["entity_abc"]
 }]
}
```

Return logs for entity *entity_abc* and any entities usedBy *entity_abc*:

```
"targetFilter":   {
 "filters":[{
    "name":"usedBy",
    "operator":"isEqual",
    "values":["entity_abc"]
 }]
}
```

Return logs for entity *entity_abc* and any entities that use *entity_abc*:

```
"targetFilter":   {
 "filters":[{
   "name":"uses",
   "operator":"isEqual",
   "values":["entity_abc"]
 }]
}
```

Use Keywords, Phrases, and Wildcards

String queries can include keywords and phrases. A keyword is a single word (for example, `database`), while a phrase refers to multiple words, enclosed in single (' ') or double (" ") quotes (for example, `'database connection'`). If you specify a keyword or a phrase in your query, then all log entries containing the specified keyword or phrase are returned after the query is run.

The Oracle Log Analytics search language also supports special pattern mapping. In other words, you can use wildcard characters, such as asterisk (`*`), question mark (`?`), and percentage (`%`), to complete keywords.

The following table lists the supported wildcard characters and provides a brief description of each.

Wildcard Character	Description
`?`	Use this character to match exactly one character of the possibilities to the keyword. For example, if you enter `host?`, then the keyword `host1` is considered to be a match, while `host.foo.bar` is not.

Wildcard Character	Description
`*` or `%`	Use either of these characters to match 0 or more characters of the possibilities, to the keyword. For example, if you enter `host*` or `host%`, then `host1` and `host.foo.bar` are considered to match the keyword. Similarly, if you enter `%host%`, then `ahostb` and `myhost` are considered to match the specified keyword.

You can specify multiple keywords. For example, `database` and `connection`. Logs containing the words `database` and `connection` (but not necessarily together) are returned. However, these words need not necessarily occur consecutively. However, by enclosing the words in quotes and including them in the query string as a phrase (`‘database connection’`, for example), then only those logs containing the phrase `‘database connection’` are returned. To see how to use multiple keywords, see Use Boolean Expressions.

When specifying a keyword or phrase, remember the following:

- Keywords and phrase strings are not case-sensitive.
- Keywords which are not enclosed within quotes must contain only alphanumeric characters, underscore (_), and wildcard characters (`*`, `%`, and `?`).
- Keyword searches where the substring could be interpreted as a separate directive should be specific within quotes. For example, to search for the string `and`, you will have to enter it within single quotes (`‘and’`) to prevent the system from picking up its Boolean meaning.

Note:

To use wildcards with the `message` field, you must also use `LIKE` or `LIKE IN`. Following are examples of using wildcards with `message`.

```
ORA-* AND message LIKE 'connection* error*'
```

```
ORA-* AND message LIKE IN ('tablesp*','connection* error*')
```

Use Comparison Operators

Comparison operators are conditions you specify to establish a relationship between a field and its value. Fields without values are considered to be null.

The following table lists the supported comparison operators and provides a brief description of each.

Comparison Operator	Description
`<`	If you use this operator in your query, then all log entries with a value, for the corresponding field, of less than the specified value are returned.

Comparison Operator	Description
`<=`	If you use this operator in your query, then all log entries with a value, for the corresponding field, of less than or equal to the specified value are returned.
`>`	If you use this operator in your query, then all log entries with a value, for the corresponding field, of *greater than* the specified value are returned.
`>=`	If you use this operator in your query, then all log entries with a value, for the corresponding field, of *greater than or equal to* the specified value are returned.
`=`	If you specify this operator in your query, then all log entries with a value, for the corresponding field, of *equal to* the specified value are returned.
`!=`	If you specify this operator in your query, then all log entries with a value, for the corresponding field, of *not equal to* the specified value are returned.

Use these operators to find logs with fields having specific values. For example, specify `Severity=’ERROR’` to search through the available logs where the value of the field `Severity` is `ERROR`. Similarly, `Severity!=NULL` returns all logs where the value of the `Severity` field is not null (in other words, where severity has been specified).

Note:

The value to the right of the comparison operator must be specified within quotes if the value is not numeric or `NULL`.

Use Boolean Expressions

The Oracle Log Analytics Search feature has the capabilities of `LIKE` and `REGEX`, as per standard conventions. Boolean Expressions can have a value of either `true` or `false`.

The following table lists the supported Boolean Expressions, along with a brief description of each.

Boolean Expression	Description
`AND`	Use this expression to view only those logs which contain both specified parameters.

Boolean Expression	Description
`NOT IN` or `IN`	Use this expression to find data which is in a specified subset of available data. For example, `'Entity Type' IN ('Database Instance','Automatic Storage Management','Listener','Cluster')` will first consider only those logs which contain `'Database Instance'`, `'Automatic Storage'`, `Listener`, or `Cluster`, and then identify those logs containing `'Entity Type'`. However, when you use `NOT IN`, then log entries with the specified keyword or phrase are returned, excluding the specified entries. For example, `'Entity Type' NOT IN ('Database Instance','Automatic Storage Management','Listener','Cluster')` first filters out log entries with `'Database Instance'`, `'Automatic Storage Management'`, `Listener`, and `Cluster`, and then returns those log entries where the value of `'Target Type'` is not one of the specified values. The reserved word `NULL` is supported with this Boolean operator.
`NOT LIKE` or `LIKE`	Use this expression to find data which either matches or does not match the specified character pattern. The character pattern is a string that can contain one or more wildcard characters.
`NOT LIKE IN` or `LIKE IN`	Similar to `[NOT] IN`, this expression allows you to use a shorthand for expressing multiple `LIKE` clauses together.
`OR`	Use this to view those logs which contain either of the specified parameters.

The Oracle Log Analytics Search language supports nesting Boolean expressions within other Boolean expressions. For example, consider the following query:

```
fatal ('order' OR host LIKE '*.oracle.com')
```

Running this query returns all logs which contain `fatal` and either contain the keyword `order` or originated from a host whose name ends with `.oracle.com`.

Understand the Search Commands

The Search Language for analyzing the logs allows you to specify what action to perform on the search results.

Commands can be either search commands or statistical commands.

Search Commands

Search commands are those commands which further filter the available log entries.

The following table lists the search commands and provides a brief description of each.

Command	Description
`addfields`	Use this command to generate aggregated data within groups identified by the `link` command. See Addfields Command.
`addinsights`	Use this command to view additional insight information in each log record. See Addinsights Command.
`bottom`	Use this command to display a specific number of results with the lowest aggregated value as determined by the specified field. See Bottom Command.
`bucket`	Use this command to group the log records into buckets based on the range of values of a field. See Bucket Command.
`classify`	Use this command to cluster properties of groups identified by the `link` command. See Classify Command.
`cluster`	Use this command to group similar log records. See Cluster Command.
`clustercompare`	Use this command to compare one cluster collection with another, and for viewing the clusters that exist exclusively in the current range versus clusters that exist exclusively in the baseline range. See Clustercompare Command.
`clusterdetails`	Use this command to return similar log records. See Clusterdetails Command.
`clustersplit`	Use this command to view the log data within a cluster for specific classify results in the tabular format. See Clustersplit Command.
`compare`	Use this command to compare properties generated by the `link` command over the comparison intervals specified. See Compare Command.
`createview`	Use this command to define a subquery to create a subset of groups identified by the `link` command. See Createview Command.
`distinct`	Use this command to remove duplicates from the returned results. See Distinct Command.
`eval`	Use this command to calculate the value of an expression and display the value in a new field. See Eval Command.
`eventstats`	Use this command to obtain overall summary statistics, optionally grouped by fields, on properties of groups identified by the `link` command. Its output will include one field for each aggregation. See Eventstats Command.
`fields`	Use this command to specify which fields to add or remove from the results. See Fields Command.
`fieldsummary`	Use this command to return data for the specified fields. See Fieldsummary Command.
`head`	Use the `head` command to display the first *n* number of results. See Head Command.

Command	Description
`highlightgroups`	Use this command to match strings or search criteria on the properties of the groups identified by the `link` command, and causes them to be highlighted in the link visualization. See Highlightgroups Command.
`highlightrows`	Use this command to match a string or a list of strings, and highlight the entire row in the Log UI. See Highlightrows Command.
`highlight`	Use this command to match a string or a list of strings, and highlight them in the Log UI. See Highlight Command.
`link`	Use this command to group log records into high level business transactions. See Link Command.
`lookup`	Use this command to invoke field value lookups. See Lookup Command.
`map`	Use this command to join a view with the groups identified by the `link` command to create new properties. See Map Command.
`nlp`	Use this command to apply natural language processing algorithms to a text field. See NLP Command.
`regex`	Use this command to filter data according to a specified regular expression. See Regex Command.
`rename`	Use this command to change the name of a field. See Rename Command.
`search`	Use this command to retrieve a specific logical expression from the available log data. See Search Command.
`searchLookup`	Use this command to retrieve contents from a lookup table. See SearchLookup Command.
`sort`	Use this command to sort logs according to specified fields. See Sort Command.
`tail`	Use this command to display the last *n* number of results. See Tail Command.
`timecluster`	Use this command to group the time-series charts together based on how similar they are to one another. See Timecluster Command.
`top`	Use this command to display a specified number of results with the highest aggregated value as determined by the specified field. See Top Command.
`where`	Use this command to calculate the value of an expression to be true or false. See Where Command.

Statistical Commands

Statistical commands perform statistical operations on the search results.

The following table lists the supported statistical commands, and provides a short description for each.

Commands	Description
`distinct`	Use this command to remove duplicate entries from the search results. See Distinct Command.
`stats`	Use this command to provide summary statistics for the search results, optionally grouped by a specified field. See Stats Command.
`timestats`	Use this command to generate data for displaying statistical trends over time, optionally grouped by a specified field. See Timestats Command.

Write Sub-Queries

Sub-queries allow the child query to provide a dynamic filter to its parent queries. Sub-queries are evaluated first, and the result is then used in the parent query.

- You can nest sub-queries inside one another as well as a particular query having multiple sub-queries at the same level.
- Sub-queries by default inherit the global time range, but you can override it using the `time between T1 and T2` syntax, if required.
- Sub-queries are restricted to return only the first 2000 matches as input to its parent. Other results are truncated.
- They have a maximum timeout of 30 seconds to complete.
- All of the fields returned by a sub-query must match the fields in the parent query by name. Otherwise, it will result in an error.
- You can use sub-queries only inside a search command.
- You can use all the commands inside a sub-query except `cluster`, `clustersplit`, `clustercompare`, `fieldsummary`, `delete`, `classify`, `highlight`, and `highlightrows`.

Examples:

- **Chart the traffic from the IP blacklist over time**:

```
[searchlookup table=ip_blacklist | distinct ip | rename ip as 'host
address'] | timestats count
```

- **List the most purchased products for the top users of an e-commerce site**:

```
'Log Source'='WLS Access Logs' status=200 action=purchase ['Log
Source'='WLS Access Logs' status=200 action=purchase | stats count by
'Host (Client Address)' | top limit=1 'Host(Client Address)' | fields -*,
'Host (Client Address)'] | lookup table=products select 'product name'
using productid | stats count, distinctcount(productId), unique('product
name') by 'Host (Client Address)'
```

- **Find Top 4 OS Process IDs with highest sum**:

  ```
  [ *|stats sum('OS Process ID') as OSprocessidSum by 'OS Process ID'
  | top 4 OSprocessidSum | fields -OSprocessidSum ] | stats count by
  'OS Process ID', 'Log Source', 'Host Name(Server)'
  ```

- **Show all the logs from the target with the most fatal severity logs**:

  ```
  * and [ Severity = fatal | stats count by Target | top limit = 1
  Count | fields -Count]
  ```

A

Command Reference

Specify commands in your query string to perform specific actions on the search results.

The first and implicit command in a query is the `search` command. This command consists of a series of keywords, and fieldname-value pairs, which identify the data that needs to be retrieved. More commands can be specified by separating them from the `search` command by using a pipe character (`|`).

The following commands are supported:

- Addfields Command
- Addinsights Command
- Bottom Command
- Bucket Command
- Classify Command
- Cluster Command
- Clustercompare Command
- Clusterdetails Command
- Clustersplit Command
- Compare Command
- Createview Command
- Distinct Command
- Eval Command
- Eventstats Command
- Fields Command
- Fieldsummary Command
- Head Command
- Highlightgroups Command
- Highlightrows Command
- Highlight Command
- Link Command
- Lookup Command
- Map Command
- NLP Command
- Regex Command
- Rename Command

- Search Command
- SearchLookup Command
- Sort Command
- Stats Command
- Tail Command
- Timecluster Command
- Timestats Command
- Top Command
- Where Command

Addfields Command

Use the `addfields` command to generate aggregated data within groups identified by the `link` command. The output of the command includes one field for each aggregation in the `stats` sub-query.

You can use `addfields` command with the run time fields that are generated using `stats`, `eventstats`, and `eval` commands.

Syntax

```
* | link <field_name> | addfields <subquery> [, <subquery>]
```

where *subquery* can be expanded as follows: `[ <logical_expression> / <boolean_expression> | <eventstats_functions> / <stats_functions> ]`

Parameters

The following table lists the parameters used in this command, along with their descriptions.

Parameter	Description
`logical_expression`, *`boolean_expression`*	Each sub-query must begin with a logical or a boolean expression to select a subset of data within each group. See Search Command and Where Command for details on the expressions.
`eventstats_functions`	The eventstats functions to apply on group properties. See Eventstats Command for the details on the available functions.
`stats_functions`	The stats functions to apply on the selected data. See Stats Command for details on the available functions.

For examples of using this command in typical scenarios, see:

- Link by Using SQL Statement as the Field of Analysis in *Using Oracle Log Analytics*
- Analyze the Time Taken Between Steps in a Transaction in *Using Oracle Log Analytics*

- Use Link Navigation Functions to Identify Events in a Database in *Using Oracle Log Analytics*

The following command returns counts based on entity name pattern for each entity type:

```
* | link 'Entity Type'
| addfields
    [ substr(Entity, 0, 3) = 'adc' | stats count as 'ADC Count' ],
    [ substr(Entity, 0, 3) = 'slc' | stats count as 'SLC Count']
```

The following command returns counts based on entity name pattern for each entity type:

```
* | link 'Entity Type'
| stats avg('Content Size') as 'Content Size', earliest(Severity) as
Severity
| addfields
    [ * | where 'Entity Type' = 'Cluster Database'
        | sort 'Content Size'
        | eventstats first('Content Size') by Severity
    ]
```

Identify the last event using the row number:

```
'Log Source' = 'Database Alert Logs' and Label != null and Entity = MyDB
| rename Entity as Database
| link span = 1minute Time, Database, Label
| sort Database, 'Start Time'
| eventstats rownum as 'Row Number' by Database
| addfields
   [ * | where Label = 'Abnormal Termination'
       | eventstats last('Row Number') as 'Crash Row'
   ]
```

Addinsights Command

Use the `addinsights` command to view additional insight information in each log record. The commands returns fields *Cluster Record Count*, *Shape Record Count*, *Shape Cluster Count*, *Potential Issue*, and *Shape ID*.

Syntax

```
addinsights
```

Fields in the Result

The following table lists the parameters used in this command, along with their descriptions.

Parameter	Description
`Cluster Record Count`	Number of log records that are similar to this one
`Shape Record Count`	Number of log records with the same trend over time

Parameter	Description
`Shape Cluster Count`	Number of Clusters with similar trend over time
`Potential Issue`	This field can have a value of `null` or `1`. A value of `1` indicates the log record contains keywords that are considered as potential issues.
`Shape ID`	A unique ID that represents the shape for this record's trend. All records with this ID will have the same trend. You can click on the cluster trend to search for similar trends.

The following command adds insights to log records with severity fatal.

```
severity = fatal | addinsights | where 'Shape ID' = -1845058765
```

Bottom Command

Use the `bottom` command to display *n* number of results (where *n* is a number you specify) with the lowest aggregated value as determined by the specified field. This command must be preceded with a `STATS` or `CLUSTER` command. When you use this command, the results of the command passed before the pipe character are sorted in ascending order, based on the field and number specified when running the query.

Syntax

```
[stats|cluster] | bottom [limit=<limit>] <field_name>
```

Parameters

The following table lists the parameters used in this command, along with their descriptions.

Parameter	Description
`field_name`	Specify the field by which you want the results to be sorted.
`limit`	Specify the number of entries you want to see. If no value is specified, then the default value of `10` is used. If you enter a value of `-1`, then all rows are returned.

The following command returns the 10 log sources with the lowest number of log entries.

```
* | stats count as cnt by Source | bottom cnt
```

The following command returns 20 targets with the fewest fatal log entries.

```
Severity = fatal | stats count as cnt by 'Entity Type', Entity |
bottom limit = 20 cnt
```

The following command returns 10 summaries with the fewest number of similar log records.

```
* | cluster | bottom Count
```

Bucket Command

Use the `bucket` command to group the log records into buckets based on the range of values of a field. The buckets can be created automatically based on the values of the field or can be specified.

Syntax

```
* | bucket [<bucket_options>] <field_name> [<ranges>]
```

Parameters

The following table lists the parameters used in this command, along with their descriptions.

Parameter	Description
`bucket_options`	**Syntax**: `[span = <size>] [maxbuckets= <maxbuckets>]` `size` specifies the size of each bucket. If the span is not specified, then the buckets will be evenly spaced between the minimum and maximum values. `maxbuckets` specifies the maximum number of buckets to create. If not specified, then the default is 10.
`field_name`	Specify the **numeric** field to use for grouping.
`ranges`	**Syntax**: `[<range>] [,<range>, ...] [, others = <others>]` `range` syntax: `[alias] = <lower> - <upper>` `others` specifies the name of the bucket that has values which don’t fit into any of the specified buckets. If not specified, the name is `others`.

The following command automatically creates a bucket of the field `Query Duration`:

```
* | bucket 'Query Duration'
```

The following command automatically creates `5` buckets of the field `Query Duration`:

```
* | bucket maxbuckets=5 'Query Duration'
```

The following command automatically creates buckets with the names `fast`, `medium`, and `slow` of the field `Query Duration`:

```
* | bucket 'Query Duration' fast=0-1000, medium=1001-5000, others=slow
```

Classify Command

Use `classify` command to cluster properties of groups identified by the `link` command. This command returns the following details and the minimum and maximum range of the properties analyzed:

- *Id* – Cluster identifier
- *Group Count* – Number of groups within a cluster
- *Percentage* - Percentage of a cluster relative to the cluster distribution
- *Distance* - Distance of a cluster relative to the cluster distribution
- *Anomaly* - Whether a cluster is an anomaly relative to the cluster distribution
- *Anomaly Baseline* - Baselines used for identifying a cluster as an anomaly

Syntax

```
* | link <field_name> | classify [<classify_options>] <field_name> [,
<field_name>] [as
    <new_field_name>]
```

Parameters

The following table lists the parameters used in this command, along with their descriptions.

Parameter	Description
`classify_options`	Syntax: `[topcount = <count>] [bottomcount = <count>]` where `count` is the maximum number of clusters to return.

For examples of using this command in typical scenarios, see:

- Rename the Fields by Editing the Query in *Using Oracle Log Analytics*
- Use Dictionary Lookup in Link in *Using Oracle Log Analytics*
- Generate Link Alerts in *Using Oracle Log Analytics*
- Visualize Time Series Data Using the Link Trend Feature in *Using Oracle Log Analytics*
- Analyze the Access Logs of Oracle WebLogic Server in *Using Oracle Log Analytics*
- Link by Using SQL Statement as the Field of Analysis in *Using Oracle Log Analytics*
- Analyze the Time Taken Between Steps in a Transaction in *Using Oracle Log Analytics*

The following command returns an analysis of severity versus count for every transaction:

```
* | link 'Transaction ID' | classify Severity, Count as 'Severity Analysis'
```

Cluster Command

Use this command to group similar log records. The cluster command uses machine learning to group log records together based on how similar they are to each other. Clustering helps significantly reduce the total number of log entries the user has to explore and easily points out the outliers. Grouped log entries are presented as *message signatures*.

Syntax

```
cluster [<field_name>,(<field_name>)*]
```

For examples of using this command in typical scenarios, see:

- Use Dictionary Lookup in Cluster in *Using Oracle Log Analytics*
- Link by Cluster in *Using Oracle Log Analytics*
- Generate Alerts for Cluster Utilities in *Using Oracle Log Analytics*

The following command performs a cluster analysis on all the fatal logs.

```
Severity = fatal | cluster
```

The following command performs a cluster analysis on all fatal logs, and returns the summary groupings in ascending order.

```
Severity = fatal | cluster | sort Count
```

Clustercompare Command

Use `clustercompare` command to compare one cluster collection with another, and for viewing the clusters that exist exclusively in the current range versus clusters that exist exclusively in the baseline range. This command returns a table with nine columns:

- *Collection* – The name of the collection where data is persisted
- *Id* – Cluster id that is unique within the collection
- *Log Source* - The source of the cluster
- *Count* - Number of log records with this signature
- *Cluster Sample* - A sample log record from the signature
- *Sample Count* - The number of samples for each pattern, may be one or more in certain cases
- *Shape* - A computed number assigned to each unique trend to group similar trends together
- *Trend* - Trend of log entries that match the pattern over time
- *Score* - A computed value assigned to each cluster used in default sorting

Syntax

```
clustercompare [timeshift = <offset> | starttime = <datetime> endtime
= <datetime>] [includetrends = [true | false]] [span = <span>]
[<baseline_query>]
```

Parameters

The following table lists the parameters used in this command, along with their descriptions.

Parameter	Description

`offset`	`offset` sets the baseline cluster collection time range as an offset of the current time range.
	Syntax: `<direction><int><timescale>`
	`direction` values: `+ | -`
	`timescale` syntax: `<sec> | <min> | <hour> | <day> | <week> | <mon>`
	• `sec` values: s | sec | secs | second | seconds
	• `min` values: m | min | mins | minute | minutes
	• `min` values: h | hr | hrs | hour | hours
	• `min` values: w | week | weeks
	• `min` values: mon | month | months

`datetime`	Explicitly sets start and end time range of baseline cluster collection.
`includetrends`	Indicates if the results should include trend data. If includeTrends is not specified a default of true is used.
`span`	Sets the size of the length of time to be used for the result histogram. Syntax: `<int><timescale>` `timescale` syntax: `<sec> | <min> | <hour> | <day> | <week> | <mon>` • `sec` values: s | sec | secs | second | seconds • `min` values: m | min | mins | minute | minutes • `min` values: h | hr | hrs | hour | hours • `min` values: w | week | weeks • `min` values: mon | month | months
`baseline_query`	Lets you specify a different search query for the baseline cluster collection.

The following command compares host1 clusters in current range with host2 clusters from 7 days ago:

```
Entity = host1 | clustercompare timeshift = -7days [ Entity = host2]
```

The following command compares clusters in current range with clusters from another period of time:

```
* | clustercompare starttime = '2018-06-07T00:00:00Z' endtime
='2018-06-14T00:00:00Z'
```

Clusterdetails Command

Use this command to look at log data within categories for specific `classify` results It enables you to expand a message signature into the individual log entries.

Syntax

```
clusterdetails collection=<collection_name> [<summary_expression>]
```

Parameters

The following table lists the parameters you can use with this command, along with their descriptions.

Parameter	Description
`collection_name`	Use this parameter to specify the collection where the log data exists. The value for this variable should either be in the format `'<string>'` or `"<string>"`.
`summary_expression`	Use this parameter to compare the ID to an expression. The value for this parameter should either be in the format `id <cmp>` or `id <in_exp>`.
`cmp`	Use this parameter as a comparison operator. The possible values for this variable include `=` and `!=`.
`in_exp`	This parameter should be in the format `[NOT] IN "(" <value> (","<value>)*")"`.

The following query returns the fatal logs included in `ID 1`, in the collection 'Fatal logs'.

```
Severity = fatal | clusterdetails collection = 'Fatal logs' id = 1
```

Clustersplit Command

Use this command to view the log data within a cluster for specific `classify` results in the tabular format.

Syntax

```
clustersplit collection=<collection_name> [<summary_expression>]
```

Parameters

The following table lists the parameters you can use with this command, along with their descriptions.

Parameter	Description
`collection_name`	Use this parameter to specify the collection where the log data exists. The value for this variable should either be in the format `'<string>'` or `"<string>"`.
`summary_expression`	Use this parameter to compare the ID to an expression. The value for this parameter should either be in the format `id <cmp>` or `id <in_exp>`.
`cmp`	Use this parameter as a comparison operator. The possible values for this variable include `=` and `!=`.
`in_exp`	This parameter should be in the format `[NOT] IN "(" <value> (","<value>)*")"`.

The resulting table on running the query has the following columns:

- **Collection**: The name of the collection where data is persisted
- **Id**: Cluster Id that is unique within the collection
- **Log Source**: The source of the cluster
- **Count**: The number of log records with this signature
- **Sample Id**: Unique identifier for the sample message
- **Sample Message**: A sample log record from the signature
- **Shape**: A computed number assigned to each unique trend to group similar trends together
- **Trend**: Trend of log entries that match the pattern over time
- **Score**: A computed value assigned to each cluster used in the default sorting
- **Facet Message Id**: Unique row identifier when splitting a cluster by facet variables
- **Variables**: Detailed information of all facet variables for each sample message
- **Document ID**: The document identifier associated with the sample message

The following query returns the fatal logs included in `ID 1`, in the collection 'Fatal logs'.

```
Severity = fatal | clustersplit collection = 'Fatal logs' id = 1
```

Compare Command

Use the `compare` command to compare properties generated by the `link` command over the comparison intervals specified.

Syntax

```
compare [fields=<field> [,<field>]*] [timeshift = <offset> [size =
<size>][count=<int>] | timerange <datetime> to <datetime> [as
<new_field_name>], ...]
```

Parameters

The following table lists the parameters used in this command, along with their descriptions.

Parameter	Description
`offset`	*`<direction><int><timescale>`* Sets comparison time range as an offset of the current time range. • `direction` syntax: + | - • `timescale` syntax: *<sec>* | *<min>* | *<hour>* | *<day>* | *<week>* | *<mon>* • `sec` syntax: s | sec | secs | second | seconds • `min` syntax: m | min | mins | minute | minutes • `hour` syntax: h | hr | hrs | hour | hours • `week` syntax: w | week | weeks • `month` syntax: mon | month | months
`size`	*`<int><timescale>`* Sets the size of the length of time of the comparison time range. • `timescale` syntax: *<sec>* | *<min>* | *<hour>* | *<day>* | *<week>* | *<mon>* • `sec` syntax: s | sec | secs | second | seconds • `min` syntax: m | min | mins | minute | minutes • `hour` syntax: h | hr | hrs | hour | hours • `week` syntax: w | week | weeks • `month` syntax: mon | month | months
`datetime`	Explicitly sets start and end of a comparison time range.

For example of using this command in typical scenarios, see:

- Compare Link Metrics Across Time in *Using Oracle Log Analytics*

The following command compares average content size of an entity from 7 days ago and from 14 days ago:

```
* | link Entity
| stats avg('Content Size') as 'Average Content Size'
| compare fields = 'Average Content Size' timeshift = -7d count = 2
```

The following command compares average content size of an entity from another period of time:

```
* | link Entity
| stats avg('Content Size') as 'Average Content Size'
| compare fields = 'Average Content Size'
        timerange = '2018-06-07T00:00:00Z' to '2018-06-14T00:00:00Z' as T1
```

Createview Command

Use the `createview` command to define a subquery to create a subset of groups identified by the link command. This virtual set of groups will in turn be used in other commands for further computations.

Syntax:

```
createview <subquery> as <view_name>
```

Parameters

The following table lists the parameters used in this command, along with their descriptions.

Parameter	Description
`subquery`	The subquery to create a subset of groups. The subquery must use the rename command to expose fields that need to made available when this view is used.
`view_name`	The name of the view.

For examples of using this command in typical scenarios, see:

- Create Sub-Groups Using Createview Command in *Using Oracle Log Analytics*

The following command creates `Application Error View` view of all the entity groups with application error:

```
* | link Entity, Label
  | createview [
    * | where Label = 'Application Error'
    | rename Entity as 'Application Error Entity'
    ] as 'Application Error View'
```

Distinct Command

Use this command to remove duplicates from the returned results.

Syntax

```
distinct <field_name> (,<field_name>)*
```

Parameters

The following table lists the parameters used with this command, along with their descriptions.

Parameter	Description
`field_name`	Use this parameter to specify the field name.

The following query returns the distinct list of entities having `ORA-00600` errors.

```
Message like '%ORA-00600%' | distinct 'Entity Type', Entity
```

Eval Command

Use the `eval` command to calculate the value of an expression and display the value in a new field.

Note:

While the `stats` command calculates statistics based on existing fields, the `eval` command creates new fields by using existing fields and arbitrary expressions.

Syntax

```
*|eval <new_field_name>=<expression>
```

Operators and Functions Available with the Command

The following table lists the operators available with the `eval` command.

Category	Example
Arithmetic Operators	`+`, `-`, `*`, `/`, `%`
Comparison Operators	`=`, `!=`, `<`, `>`, `<=`, `>=`
Logical Operators	`and`, `or`, `not`
Conditional Operators	`if(<expression>,<expression>,<expression>)`
Multiple Comparison Operators	`in`, `not in`

The following table lists the functions available with the `eval` command.

Category	Example
String Functions	• `concat(String, String)` • `indexof (String, String [,int])` • `length(String)` • `literal(String)` • `lower(String)` • `ltrim(String, Character)` • `replace(String, String, String)` • `rtrim(String, Character)` • `substr(String, int [, int])` • `todate(String [, format])` • `toduration(String)` • `tonumber(String)` • `trim(String)` • `trim(String, Character)` • `upper(String)` • `urldecode(String)` • `url(String [, Name [, Parameter]])` See `url` Function Details.
Numeric Functions	• `abs(number)` • `ceil(number)` • `floor(number)` • `formatduration(number)` • `max(number, number)` • `min(number, number)` • `power(number, int)` • `round(number, int)` • `sqrt(number)` • `tostring(number)` • `unit(number, unit)` See Supported Types for the `unit` function and Supported Currency Types in the `unit` Function.
Date Functions	• `dateadd(date, property, amount)` • `dateset(date, property, value [, property, value])` • `formatdate(date [,format])` • `now()`
Network Functions	`cidrmatch(String, String)`

Note:

- For the `concat()` function, you can input numeric data types like integer, float, or long. The numeric fields with be automatically converted to the corresponding string values.
- You can use `||` to concatenate *n* number of inputs. Here too, you can input numeric data types which will be automatically converted to the corresponding string values.

Parameters

The following table lists the parameters used in this command, along with their descriptions.

Parameter	Description
`new_field_name`	Specify the name of the field where the calculated value of the expression is to be displayed.
`expression`	Specify the expression for which the value needs to be calculated.

Supported Types for the `unit` function

Unit Names:

- `BYTE`
- `KILOBYTE | KB`
- `MEGABYTE | MB`
- `GIGABYTE | GB`
- `TERABYTE | TB`
- `PETABYTE | PB`
- `EXABYTE | EB`
- `MICRO | µs`
- `MILLISECOND | MS`
- `S | SEC | SECS | SECOND | SECONDS`
- `M | MIN | MINS | MINUTE | MINUTES`
- `H | HR | HRS | HOUR | HOURS`
- `D | DAY | DAYS`
- `W | WEEK | WEEKS`
- `MON | MONTH | MONTHS`
- `Y | YR | YRS | YEAR | YEARS`
- `PERCENT | PCT`

Supported Currency Types in the `unit` Function

Specify the currency unit using the following format:

```
eval <New Field> = unit(<Field>, currency_<ISO-4217 Code>)
eval <New Field> = unit(<Field>, currency_<ISO-4217 Code>_k)
eval <New Field> = unit(<Field>, currency_<ISO-4217 Code>_m)
eval <New Field> = unit(<Field>, currency_<ISO-4217 Code>_b)
```

The suffixes `_k`, `_m` and `_b` are used to indicate the currency in thousands, millions or billions, respectively. For a full list of currency codes, see *ISO Standards*.

`NLS_Territory`	Currency
AFGHANISTAN	AFN
ALBANIA	ALL
ALGERIA	DZD
AMERICA	USD
ANGOLA	AOA
ANTIGUA AND BARBUDA	XCD
ARGENTINA	ARS
ARMENIA	AMD
ARUBA	AWG
AUSTRALIA	AUD
AUSTRIA	EUR
AZERBAIJAN	AZN
BAHAMAS	BSD
BAHRAIN	BHD
BANGLADESH	BDT
BARBADOS	BBD
BELARUS	BYN
BELGIUM	EUR
BELIZE	BZD
BERMUDA	BMD
BOLIVIA	BOB
BOSNIA AND HERZEGOVINA	BAM
BOTSWANA	BWP
BRAZIL	BRL
BULGARIA	BGN
CAMBODIA	KHR
CAMEROON	XAF
CANADA	CAD
CAYMAN ISLANDS	KYD
CHILE	CLP
CHINA	CNY
COLOMBIA	COP
CONGO BRAZZAVILLE	XAF

`NLS_Territory`	Currency
CONGO KINSHASA	CDF
COSTA RICA	CRC
CROATIA	HRK
CURACAO	ANG
CYPRUS	EUR
CZECH REPUBLIC	CZK
DENMARK	DKK
DJIBOUTI	DJF
DOMINICA	XCD
DOMINICAN REPUBLIC	DOP
ECUADOR	USD
EGYPT	EGP
EL SALVADOR	SVC
ESTONIA	EUR
ETHIOPIA	ETB
FINLAND	EUR
FRANCE	EUR
FYR MACEDONIA	MKD
GABON	XAF
GEORGIA	GEL
GERMANY	EUR
GHANA	GHS
GREECE	EUR
GRENADA	XCD
GUATEMALA	GTQ
GUYANA	GYD
HAITI	HTG
HONDURAS	HNL
HONG KONG	HKD
HUNGARY	HUF
ICELAND	ISK
INDIA	INR
INDONESIA	IDR
IRAN	IRR
IRAQ	IQD
IRELAND	EUR
ISRAEL	ILS
ITALY	EUR
IVORY COAST	XOF
JAMAICA	JMD
JAPAN	JPY
JORDAN	JOD
KAZAKHSTAN	KZT
KENYA	KES

NLS_Territory	**Currency**
KOREA	KRW
KUWAIT	KWD
KYRGYZSTAN	KGS
LAOS	LAK
LATVIA	EUR
LEBANON	LBP
LIBYA	LYD
LIECHTENSTEIN	CHF
LITHUANIA	EUR
LUXEMBOURG	EUR
MACAO	MOP
MALAWI	MWK
MALAYSIA	MYR
MALDIVES	MVR
MALTA	EUR
MAURITANIA	MRU
MAURITIUS	MUR
MEXICO	MXN
MOLDOVA	MDL
MONTENEGRO	EUR
MOROCCO	MAD
MOZAMBIQUE	MZN
MYANMAR	MMK
NAMIBIA	NAD
NEPAL	NPR
NEW ZEALAND	NZD
NICARAGUA	NIO
NIGERIA	NGN
NORWAY	NOK
OMAN	OMR
PAKISTAN	PKR
PANAMA	PAB
PARAGUAY	PYG
PERU	PEN
PHILIPPINES	PHP
POLAND	PLN
PORTUGAL	EUR
PUERTO RICO	USD
QATAR	QAR
ROMANIA	RON
RUSSIA	RUB
SAINT KITTS AND NEVIS	XCD
SAINT LUCIA	XCD
SAUDI ARABIA	SAR

`NLS_Territory`	Currency
SENEGAL	XOF
SERBIA	RSD
SIERRA LEONE	SLL
SINGAPORE	SGD
SLOVAKIA	EUR
SLOVENIA	EUR
SOMALIA	SOS
SOUTH AFRICA	ZAR
SOUTH SUDAN	SSP
SPAIN	EUR
SRI LANKA	LKR
SUDAN	SDG
SURINAME	SRD
SWAZILAND	SZL
SWEDEN	SEK
SWITZERLAND	CHF
SYRIA	SYP
TAIWAN	TWD
TANZANIA	TZS
THAILAND	THB
THE NETHERLANDS	EUR
TRINIDAD AND TOBAGO	TTD
TUNISIA	TND
TURKEY	TRY
TURKMENISTAN	TMT
UGANDA	UGX
UKRAINE	UAH
UNITED ARAB EMIRATES	AED
UNITED KINGDOM	GBP
URUGUAY	UYU
UZBEKISTAN	UZS
VENEZUELA	VES
VIETNAM	VND
YEMEN	YER
ZAMBIA	ZMW
ZIMBABWE	ZWL

`url` Function Details

The syntax for the `url` function:

```
url(String, Name, Parameter)
```

Name and *Parameter* values are optional.

- *String*: This can be a URL or one of the predefined short names. For example:

  ```
  eval Link = url('https://www.oracle.com')
  ```

- *Name*: Optional Name for the URL. For example:

  ```
  eval Link = url('https://www.oracle.com', 'Oracle Home Page')
  ```

- *Parameter*: Optional parameter if a short-cut is used for *String*. For example:

  ```
  eval Link = url('tech', 'Search Oracle', 'ORA-600')
  ```

For examples of using this command in typical scenarios, see:

- Rename the Fields by Editing the Query in *Using Oracle Log Analytics*
- Histogram Chart Options in *Using Oracle Log Analytics*
- Visualize Time Series Data Using the Link Trend Feature in *Using Oracle Log Analytics*
- Generate Charts with Virtual Fields in *Using Oracle Log Analytics*
- Link by Using SQL Statement as the Field of Analysis in *Using Oracle Log Analytics*
- Analyze the Time Taken Between Steps in a Transaction in *Using Oracle Log Analytics*
- Use Link Navigation Functions to Identify Events in a Database in *Using Oracle Log Analytics*
- Use the Currency Symbols in Your Log Analysis in *Using Oracle Log Analytics*

Following are some examples of the `eval` command.

```
*|eval newField = 'foo'
```

```
*|eval newField = 123
```

```
*|eval newField = upper(Target)
```

```
*|eval newField = length('hello world')
```

```
*|eval newField = replace('aabbcc', 'bb', 'xx')
```

```
*|eval newField = concat(host, concat (':', port))
```

```
*|eval newField = host || ':'|| port
```

```
*|eval newField = url('Destination URL')
```

```
*|eval newField = substr('aabbcc', 2, 4)
```

```
*|eval newField = round(123.4)
```

```
*|eval newField = unit('Content Size', KB)
```

```
eval 'File Size (bytes)' = unit('File Size', 'byte')
```

```
eval 'File Size (KB)' = unit('File Size'/1024, 'kb')
```

```
eval 'File Size (MB)' = unit('File Size'/(1024*1024), 'mb')
```

```
eval 'Time Taken (Sec)' = unit('Time Taken (ms)'/1000, 'SEC')
```

```
*|eval newField = floor(4096/1024)+Length
```

```
*|eval newField = if (max(Length)(Target), length(Severity)) <= 20, 'OK',
'ERROR')
```

```
*|eval newField =
urldecode('http%3A%2F%2Fexample.com%3A893%2Fsolr%2FCORE_0_0%2Fquery')
```

```
*|eval newField = 'Host Name (Destination)' in (host1, host2)
```

The following example compares the IP addresses in the field `srvrhostip` to a subnet range.

```
*|eval newField = if (cidrmatch(srvrhostip, '192.0.2.254/25') = 1,
'local', 'not local')
```

The following example returns the string “Target”.

```
*|eval newField = literal(Target)
```

The following example removes the spaces and tabs from both the ends.

```
*|eval newField = trim(Label)
```

The following example removes the matching character from both the ends.

```
*|eval newField = trim('User Name',h)
```

The following example removes the matching character from the left end.

```
*|eval newField = ltrim('Error ID',0)
```

The following example removes the matching character from the right end.

```
*|eval newField = rtrim('OS Process ID',2)
```

The following example sets the field `date` to `Start Date` and defines the format of the date as `MM/dd/yyyy HH:mm`.

```
*|eval date = toDate('Start Date', 'MM/dd/yyyy HH:mm')
```

The following example sets the value of the field `duration` to `1.30`.

```
*|eval duration = toduration("1.30")
```

The following example sets the value of the field `duration` to a numerical value which is the difference of `End Time` and `Start Time`.

```
*|eval duration = formatDuration('End Time' - 'Start Time')
```

The following examples illustrate the use of date functions.

```
*| eval lastHour = dateAdd(now(), hour, -1)
*| eval midnight = dateSet(now(), hour, 0, minute, 0, sec, 0, msec, 0)
*| eval timeOnly = formatDate(now(), 'HH:mm:ss')
*| eval now = now()
```

The following example sets the value of the field `newField` with the position of `.com` in the `uri` string.

```
*|eval newField = indexOf(uri, '.com')
```

You can use the md5, sha1, and sha256 hash functions with the eval command to filter log data. The following example sets the value of the field `user` with the value `sha1("jane")`.

```
*|eval user = sha1("jane")
```

A field with a size or duration type **unit** would be used to format the values in the Link Analyze chart, addfields histograms and the Link Table:

```
'Log Source' = 'FMW WebLogic Server Access Logs'
| link span = 5minute Time, Server
| stats avg('Duration')     as 'Raw Avg. Duration'
        avg('Content Size') as 'Raw Avg. Transfer Size'
| eval 'Average Duration'      = unit('Raw Avg. Duration', ms)
| eval 'Average Transfer Size' = unit('Raw Avg. Transfer Size', byte)
| classify 'Start Time', 'Average Duration',
          'Average Transfer Size' as 'Response Time vs. Download Sizes'
```

Mark a field as containing US Dollars, thousands of US Dollars, millions of US Dollars, or billions of US Dollars, respectively:

```
| eval 'Amount in USD' = unit('Sales Price', usd)
| eval 'Amount in Thousands (USD)' = usd('Quarterly Sales', usd_thousand)
| eval 'Amount in Millions (USD)' = usd('Annual Profit', usd_million)
| eval 'Amount in Billions (USD)' = usd('Annual Sales', usd_billion)
```

Eventstats Command

Use the `eventstats` command to obtain overall summary statistics, optionally grouped by fields, on properties of groups identified by the `link` command. Its output will include one field for each aggregation.

Note:

The *trend* aggregate operator is not permitted with `eventstats`.

Syntax

```
eventstats <stats_function> (<field_name>) [as <new_field_name>] [,
<stats_function> (<field_name>) [as <new_field_name>]]* [by <field_name> [,
<field_name>]*]
```

Parameters

The following table lists the parameters used with this command, along with their descriptions.

Parameter	Description
`field_name`	Use this parameter to specify the field according to which you want the results to be grouped.
`new_field_name`	Use this parameter to specify the new name for the field after applying the `stats` command.

Functions

The following table lists the functions available with this command, along with their examples.

Function	Examples
Values Lists the first 10 values for a particular field with optional delimiter. Can be applied to any field data type.	`values(field_name)` • `values(Label)` • `values(Severity)` • `values('Client Host City')` • `values('Client Host Country')` • `values('Client Host Continent')`
Unique Lists the first 10 unique values for a particular field with optional delimiter. Can be applied to any field data type.	`unique(field_name)` • `unique(Label)` • `unique(Severity)` • `unique('Client Host City')` • `unique('Client Host Country')` • `unique('Client Host Continent')`
Earliest Return the eldest non-null value for the specified field. Null will be returned if field is completely empty for a particular queries results.	`earliest(field_name)` • `earliest('OS Process ID')`
Latest Return the most recent non-null value for the specified field. Null will be returned if field is completely empty for a particular queries results.	`latest(field_name)` • `latest('Error ID')`
Average **Note**: This function is supported only for numeric fields.	`avg(field_name)` • `avg('Content Size')`

Function	Examples
Count **Note**: This function uses semantics similar to sql; that is, count returns the count for all rows; however, count(field) returns the count for all rows where field is not null.	`count(field_name)` • `count(Source)`
Distinct Count	`distinctcount(field_name)` • `distinctcount(Severity)`
Maximum **Note**: This function is supported only for numeric fields.	`max(field_name)` • `max('Content Size')`
Median **Note**: This function is supported only for numeric fields.	`median(field_name)` • `median('Content Size')`
Minimum **Note**: This function is supported only for numeric fields.	`min(field_name)` • `min('Content Size')`
Percentage **Note**: This function is supported only for numeric fields.	`pct(field_name, n)` • `pct('Content Size', 90)`
Sum **Note**: This function is supported only for numeric fields.	`sum(field_name)` • `sum('Content Size')`
Standard Deviation **Note**: This function is supported only for numeric fields.	`stddev(field_name)` • `stddev('Content Size')`

The following table lists the functions that are unique to this command, along with their examples.

Function	Examples
first Retrieves property value from the first row, as defined by the current sort order, within the retrieved result.	**Syntax**: `first(field_name)` • `| eventstats first('Content Size')`
last Retrieves property value from the last row, as defined by the current sort order, within the retrieved result.	**Syntax**: `last(field_name)` • `| eventstats last('Content Size')`
nthval Retrieves property value from the *n*th row, as defined by the current sort order, within the retrieved result.	**Syntax**: `nthval(field_name, n)` • `| eventstats nthval('Content Size', 2)`

Function	Examples
lag Retrieves property value from a row at a given offset prior to the current row. Default offset is 1.	**Syntax**: `lag(field_name)` • `| eventstats lag('Content Size')`
lead Retrieves property value from a row at a given offset after the current row. Default offset is 1.	**Syntax**: `lead(field_name)` • `| eventstats lead('Content Size')`
rownum Assigns a unique number sequentially, starting from 1, as defined by the current sort order to each row within the retrieved result.	**Syntax**: `rownum` • `| eventstats rownum`
peak Retrieves property value with peak magnitude, within the retrieved result. A higher value of magnitude indicates larger absolute values.	**Syntax**: `peak` • `| eventstats peak('Content Size')`
peakscore Retrieves property value with normalized peak score between 0 and 1, within the retrieved result. The score can be used to compare the peaks, with the highest peak getting 1 as the score, and all other values between 0 and 1.	**Syntax**: `peakscore` • `| eventstats peakscore('Content Size')`
valley Retrieves property value with valley magnitude, within the retrieved result. A lower value of magnitude indicates smaller absolute values.	**Syntax**: `valley` • `| eventstats valley('Content Size')`
valleyscore Retrieves property value with normalized valley score between 0 and 1, within the retrieved result. The score can be used to compare the valleys, with the least valley getting 0 as the score, and all other values between 0 and 1.	**Syntax**: `valleyscore` • `| eventstats valleyscore('Content Size')`

For examples of using this command in typical scenarios, see:

- Second Level Aggregation Using Eventstats Command in Link in *Using Oracle Log Analytics*
- Use Link Navigation Functions to Identify Events in a Database in *Using Oracle Log Analytics*

Use the **peak** and **peakscore** functions to analyze sequential data in Link Visualization for peak magnitude and a normalized score between 0 and 1. For example, the following query highlights the highest peaks in user response time, using the Duration field from the Access Logs Log Source:

```
'Log Source' = 'FMW WebLogic Server Access Logs'
| link span = 5minute Time, Server
| stats avg(Duration) as 'Avg. Duration'
| sort Server, 'Start Time'
| eventstats peak('Avg. Duration')      as 'Peak Magnitude',
             peakscore('Avg. Duration') as 'Peak Score' by Server
| highlightgroups priority = high
  [ * | where 'Peak Score' > 0.9 ] as 'High Response Time - Needs Attention'
```

This feature can be used for searching and grouping of time series data, like identifying all Out of Memory events that happened after a spike.

Group all the fatal logs by transaction, and get the overall average elapsed time across all the groups:

```
severity = fatal | link 'Transaction ID' | stats avg('Elapsed Time
(System)') as 'Average Elapsed Time' | eventstats avg('Average Elapsed
Time') as 'Overall Average Elapsed Time'
```

Group all the fatal logs by entity type and transaction, and get the overall average elapsed time across all the groups with the same entity type:

```
severity = fatal | link 'Entity Type', 'Transaction ID' | stats avg('Elapsed
Time (System)') as 'Average Elapsed Time' | eventstats avg('Average Elapsed
Time') as 'Overall Average Elapsed Time' by 'Entity Type'
```

In the Link visualization, add a number to each row:

```
'Log Source' = 'Database Alert Logs' and Label != null and Entity = MyDB
| rename Entity as Database
| link span = 1minute Time, Database, Label
| sort Database, 'Start Time'
| eventstats rownum as 'Row Number' by Database
```

Identify the last event using the row number:

```
'Log Source' = 'Database Alert Logs' and Label != null and Entity = MyDB
| rename Entity as Database
| link span = 1minute Time, Database, Label
| sort Database, 'Start Time'
| eventstats rownum as 'Row Number' by Database
| addfields
   [ * | where Label = 'Abnormal Termination'
       | eventstats last('Row Number') as 'Crash Row'
   ]
```

Identify the previous and next events of a selected event:

```
'Log Source' = 'Database Alert Logs' and Label != null and Entity =
MyDB
| rename Entity as Database
| link span = 1minute Time, Database, Label
| sort Database, 'Start Time'
| addfields
   [ *
      | where Label != null
      | eventstats lag(Label) as 'Previous Event',
                   lead(Label) as 'Next Event'
   ]
```

Fields Command

Use this command to specify which files to add or remove from the retrieved results, based on the field names.

Note:

`Original Log Content` is added by default, if no `fields` command is specified. The default field can be excluded if necessary. If the default field is excluded, and no other field is specified, then an empty response with just the matching number of results available is returned, unless that is explicitly excluded, as well.

Syntax

```
fields [+|-] <field_name> (,[+|-]<field_name>)*
```

Parameters

The following table lists the parameters and variables used with this command, along with their descriptions.

Parameter	Description
`field_name`	Use this variable to specify the field from or to which files are to be added.

For examples of using this command in typical scenarios, see:

- Link by Cluster in *Using Oracle Log Analytics*
- Link by Using SQL Statement as the Field of Analysis in *Using Oracle Log Analytics*
- Analyze the Time Taken Between Steps in a Transaction in *Using Oracle Log Analytics*

- Second Level Aggregation Using Eventstats Command in Link in *Using Oracle Log Analytics*

The following query returns a list of logs, with their timestamp, target, target type, and severity.

```
* | fields Time, Target, 'Target Type', Severity
```

Fieldsummary Command

Use this command to return data for the specified fields.

Syntax

```
fieldsummary [<fieldsummary_options>] <field_name> (,<field_name>)*
```

where the syntax for <fieldsummary_options> is:

```
[maxvalues = <limit>] [includenulls = [true|false]] [includetrends=[true|
false]]
```

Parameters and Variables

The following table lists the parameters and variables used in this command, along with their descriptions.

Parameter / Variable	Description
<maxvalues>	Use this option to specify the number of distinct values you want to see. If no value is specified for this variable, then the default of 100 is assumed. Set the value of this variable to —1 to view all distinct values.
<includenulls>	Set this option to **true** to view a null value of each field in addition to *maxvalues* number of non-null values. The default of **false** is assumed, in which case, you can view *maxvalues* number of non-null values for each field.
<includetrends>	Set this option to **false** to avoid viewing the trend data with the result of the command. The default value is **true**.

For each distinct value, this query returns the following fields:

- **field**: The field name
- **value**: The value of the field
- **count**: The number of times the specified distinct value occurs
- **trend**: Trend of log entries that match the pattern over time

The following query returns the summaries for the entity type and severity fields for all fatal logs.

```
Severity='fatal' | fieldsummary maxvalues = 10 'Entity Type', Severity
```

Head Command

Use the `head` command to display the first *n* number of results.

Syntax

```
head [limit=<limit>]
```

Parameters

The following table lists the parameters used in this command, along with their descriptions.

Parameter	Description
`limit`	Specify the number of entries you want to see. If no value is specified, then the default value of `10` is used.

For examples of using this command in typical scenarios, see:

- Use Cluster Compare Utility in *Using Oracle Log Analytics*

The following command returns the first 10 results with the severity of fatal.

```
Severity = fatal | head limit = 10
```

Highlightgroups Command

Use the `highlightgroups` command to match strings or search criteria on the properties of the groups identified by the link command, and highlight them in the link visualization.

Syntax

```
highlightgroups [<highlightgroups_options>] [<keyword_expression> [,
<keyword_expression>]*] [<subquery>] [as <new_field_name>]
```

Parameters

The following table lists the parameters used in this command, along with their descriptions.

Parameter	Description
`highlightgroups_options`	**Syntax**: *[color = red | green | blue | yellow | <hex>]* *[priority = high | medium | low]* `color`: The color to use for highlighting. `priority`: The priority assigned to the highlighted groups. If the color is not specified, then the priority is used to assign a default matching color. If priority and color are both not specified, then a default color would be used for each highlight.
`keyword_expression`	**Syntax**: *<string> | "<string>" | '<string>'* Keywords or quoted phrases to match.
`subquery`	The subquery to identify the groups.
`new_field_name`	The new name of the field.

For examples of using this command in typical scenarios, see:

- Search and Highlight Link Groups in *Using Oracle Log Analytics*

The following command highlights post request groups in yellow color:

```
* | link Request | highlightgroups color = yellow post as 'Post Requests'
```

The following command highlights entity groups with large total content size in red color:

```
* | link Entity | stats sum('Content Size') as 'Content Size' |
highlightgroups color = red [ * | where 'Content Size' > 2000000000 ] as
'Large Content'
```

Highlightrows Command

Use this command to match a string or a list of strings, and highlight the entire row in the Log UI.

Syntax

```
highlightrows <keyword_expression> [,<keyword_expression>]*
```

Parameters and Variables

The following table lists the parameters and variables used in this query, along with their descriptions.

Parameter / Variable	Description
<keyword_expression>	Use this variable to specify the string or the list of strings to be matched by running this query. Permitted values must be in the formal '*<string>*', *<string>*, or "*<string>*".

The following query highlights rows containing the keyword ERROR.

```
highlightrows ERROR
```

The following query returns all fatal logs, and highlights those rows containing the phrase `database connection`.

```
severity='fatal' | highlightrows 'database connection'
```

Highlight Command

Use this command to match a string or a list of strings, and highlight them in the Log UI.

Syntax

```
highlight [<highlight_options>] <keyword_expression>
[,<keyword_expression>]*
```

Parameters and Variables

The following table lists the parameters and variables used in this query, along with their descriptions.

Parameter / Variable	Description
<highlight_options>	Use this option to specify the color to use for highlighting the strings. The default value is `yellow`. `[color = ` *`red`* ` | ` *`green`* ` | ` *`blue`* ` | yellow]`
<keyword_expression>	Use this variable to specify the string or the list of strings to be matched by running this query. Permitted values must be in the formal '*<string>*', *<string>*, or "*<string>*".

The following query highlights the word `ERROR` in `red` color.

```
* | highlight color = red ERROR
```

The following query returns all fatal logs, and highlights the phrase `database connection` in `green` color.

```
severity = fatal | highlight color = green 'database connection'
```

The following query returns all fatal logs, highlights the phrase `database connection` in `green` color, and highlights the word `connection` in `yellow`.

```
severity = fatal | highlight color = green 'database connection' |
highlight connection
```

Link Command

Use `link` command to group log records into high level business transactions. This command returns the link by fields along with the following details:

- *Collection* – The name of the collection where data is persisted
- *Id* – Group id that is unique within the collection
- *Count* – Number of log records within a group
- *Start Time* – Earliest timestamp of log records within a group
- *End Time* – Latest timestamp of log records within a group
- *Group Duration* – Time duration of log records within a group

Syntax

```
link [<link_options>] <fieldName> [, <fieldName>), ...]
```

where `link_options` can be expanded as `[includenulls = [true|false]] [includetrends = [true|false]] [span = <span>]`.

Parameters

The following table lists the parameters used in this command, along with their descriptions.

Parameter	Description
`includenulls`	Indicates if the results should include log records where the link by fields are null. The default value is `false`.
`includetrends`	Indicates if the results should include trend data. The default value is `true`.
`span`	Sets the length of time to be used for the result histogram. Syntax: `<int><timescale>`. `timescale` syntax: `<sec> | <min> | <hour> | <day> | <week> | <mon>` • `sec` values: s | sec | secs | second | seconds • `min` values: m | min | mins | minute | minutes • `min` values: h | hr | hrs | hour | hours • `min` values: w | week | weeks • `min` values: mon | month | months

For examples of using this command in typical scenarios, see:

- Rename the Fields by Editing the Query in *Using Oracle Log Analytics*
- Use Dictionary Lookup in Link in *Using Oracle Log Analytics*
- Generate Link Alerts in *Using Oracle Log Analytics*
- Histogram Chart Options in *Using Oracle Log Analytics*
- Link by Cluster in *Using Oracle Log Analytics*
- Link by Using SQL Statement as the Field of Analysis in *Using Oracle Log Analytics*

- Analyze the Time Taken Between Steps in a Transaction in *Using Oracle Log Analytics*
- Use Link Navigation Functions to Identify Events in a Database in *Using Oracle Log Analytics*

The following command groups all the fatal logs by transaction:

```
severity = fatal | link 'Transaction ID'
```

The following command groups all the fatal logs by transaction, and gets the average elapsed time of log records within each group:

```
severity = fatal | link 'Transaction ID' | stats avg('Elapsed Time
(System)') as 'Average Elapsed Time'
```

Lookup Command

Use the `lookup` command to invoke field value lookups.

Syntax

```
lookup table=<lookupTable>[<lookup_options>] select <outputFields>
using <inputFields>
```

Parameters

The following table lists the parameters used in this command, along with their description.

Parameter	Description
`outputFields`	Syntax: *`<lookup_field> [as <log_field>] [,<lookup_field> [as <log_field>] ]*`* List of one or more fields in the lookup table that should be copied to the matching log field(s), indexed or virtual.
`lookupTable`	Name of the lookup table.
`lookup options`	Syntax: *`[maxmatches= <value>] [default=<value>]`* `maxmatches`: The maximum number of possible matches to be returned for a given lookup key. The first `n` matches, in file order, are returned. Valid values: 1-1000, inclusive. Default value is 1. `default`: The default value to be used for all output fields should a match not be found for a given lookup key. By default, this is null.

Parameter	Description
`inputFields`	Syntax: `<log_field> [= <lookup_field>] [,<log_field> [= <lookup_field>] ]*` List of one or more fields in the lookup table to match against the logs. The log field name, indexed or virtual, must be specified if different than the lookup's field name.

For examples of using this command in typical scenarios, see:

- Use Dictionary Lookup in Cluster in *Using Oracle Log Analytics*
- Use Dictionary Lookup in Link in *Using Oracle Log Analytics*

The following example shows how to annotate log records that contain ORA error code with the error’s description and severity when the lookup fields are the names of existing indexed fields.

```
* | lookup table=OraErrorCodes select description as errtxt, severity as
sevlvl using 'Error Id'=error_id
```

The following example shows how to add user group information listing no more than 5 groups.

```
* | lookup table=UserGroups maxmatches=5 select group using usrid
```

The following example shows how to annotate log records with the information from multiple lookups.

```
* | lookup table=DnsLookup select client_host using client_ip | lookup
table=AccountLookup select acct_region using acct_id
```

The following example shows how to perform two lookups using the same lookup table, but each lookup is done using different fields.

```
* | lookup table=MyLookup select B using A | lookup table=MyLookup select D
using C
```

The following example shows how to look up a value in one lookup table and then use a returned field value to do a lookup using a second lookup table.

```
* | lookup table=FirstLookup select Y using X | lookup table=SecondLookup
select Z using Y
```

Map Command

Use the `map` command to join a view that was created using the createview command, with the groups identified by the link command to create new properties.

Syntax

```
map <subquery> using <view_name>
```

Parameters

The following table lists the parameters used in this command, along with their descriptions.

Parameter	Description
`subquery`	The subquery to describe the new properties to associate with the groups. It has two parts: • A where clause that is evaluated for each group: The `where` clause can use fields from the current group, as well as fields from a view created using the createview command. This enables searching across or joining of two groups • An `eval` statement: The fields created by the eval statement are set on the groups that matched the `where` clause. A group will be matched only once.
`view_name`	The name of the view to use in the `map` command.

For examples of using this command in typical scenarios, see:

- Join Multiple Groups Using the Map Command in *Using Oracle Log Analytics*

The following command creates a property to mark all the entity groups with application error:

```
* | link Entity, Label
  | createview [
    * | where Label = 'Application Error'
    | rename Entity as 'Application Error Entity'
    ] as 'Application Error View'
  | map [ * | where Entity = 'Application Error Entity'
    | eval 'Has Issue' = Yes
    ] using 'Application Error View'
```

NLP Command

Use the `nlp` command to apply natural language processing algorithms to a text field.

Syntax

```
nlp [<nlp_options>] <cluster_function> | <keywords_function> [as
<new_field_name>]
```

Parameters

The following table lists the parameters used in this command, along with their descriptions.

Parameter	Description
`nlp_options`	**Syntax**: `[category = <category>] [wordcount = <count>] [table = <dictionary>] [similarity = <similarity_value>]`
`cluster_function`	**Syntax**: `cluster(<field>)` Generate semantic clusters from a text field.
`keywords_function`	**Syntax**: `keywords(<field>)` Extract keywords from a text field.

For examples of using this command in typical scenarios, see:

- Semantic Clustering Using Natural Language Processing in *Using Oracle Log Analytics*
- Cluster Kernel Errors in Linux Syslog Logs in *Using Oracle Log Analytics*
- Cluster the Database Alert Logs in *Using Oracle Log Analytics*

Generate semantic clusters from `Cluster Sample` text field. Extract up to 3 verbs from each `Cluster Sample`, and form a cluster only when there is a 75% match. If the table parameter is omitted, then the default dictionary *NLP General Dictionary* is used.

```
* | link cluster()
    | nlp category = verb wordcount = 3
          table = 'My NLP Dictionary'
          similarity = 0.75 cluster('Cluster Sample')
          as 'Semantic Cluster ID'
```

Extract keywords from `Cluster Sample` text field:

```
* | link cluster()
| nlp keywords('Cluster Sample') as 'Cluster Keywords'
```

Regex Command

Use the `regex` command to filter data according to a specified regular expression.

Syntax

```
regex <field> [=|!= <regular expression>] | [IN | NOT IN (<regular
expression> [(,<regular expression>)*])]
```

Parameters

The following table lists the parameters used in this command, along with their descriptions.

Parameter	Description
`field`	Specify the field to be analyzed.
`regular expression`	Specify the regular expression.

Consider the following set of sample records representing the names of entities to run a few example regex queries:

```
slc07cuc
slc07ptt
slc07ptt:8452
stuyb43
stuyb43:1831
```

Filter Requirement	Example Regex Command	Search Result
To represent a single character using `.`	`* | regex 'entity' = 'slc07ptt:845.'| distinct entity`	`slc07ptt:8452`
To detect one or more matches using the character `+`	`* | regex 'entity' = 'slc07ptt.+'|distinct entity`	`slc07ptt:8452`
To detect zero or more matches using the character `*`	`* | regex 'entity' = 'slc07ptt.*'|distinct entity`	`slc07ptt` `slc07ptt:8452`
To detect zero or one match using the wildcard character `?`	`* | regex 'entity' = 'slc07ptt?'|distinct entity`	`slc07ptt`
To specify the minimum and maximum results from the query	`* | regex 'entity' = 'slc07p{1,2}'|distinct entity`	`slc07ptt` `slc07ptt:8452`
To provide alternate options for a specific character	`* | regex 'entity' = 'slc07pt(T|t)'|distinct entity`	`slc07ptt`
To specify a complement using the `~` character	`* | regex 'entity' = 'slc~c'|distinct entity`	`slc07cuc`
To specify a numeric range using the interval option `<>`	`* | regex 'entity' = 's.*<1-43>.*'|distinct entity`	`stuyb43` `stuyb43:1831`
To join two patterns such that both of them match, use the intersection option `&`	`* | regex 'entity' = '.*43&.*tu.'|distinct entity`	`stuyb43` `stuyb43:1831`
To match any string in its entirety using the `@` option	`* | regex 'entity' = '@'|distinct entity`	`slc07cuc` `slc07ptt` `slc07ptt:8452` `stuyb43` `stuyb43:1831`
To find the records by filtering out the specific options using the not equal character `!=`	`* | regex 'entity' = 'slc07.+'|distinct entity`	`stuyb43` `stuyb43:1831`

Filter Requirement	Example Regex Command	Search Result
To find records that contain the characters 2 and 5, specify the boolean expression IN	`* | regex 'entity' in ('.*2.*', '.*5.*')| distinct entity`	slc07ptt:8452
To find records that don't contain the characters 1 and 2, specify the boolean expression NOT IN	`* | regex 'entity' not in ('.*1.*', '.*2.*')| distinct entity`	slc07cuc slc07ptt stuyb43
To specify multiple regex queries	`* | regex 'entity' = '[^1]+' | regex 'entity' = '[^5]+' | distinct entity`	slc07cuc slc07ptt stuyb43
To find the records by specifying the character class that negates the presence of the characters from1 to 6 by using the option ^	`* | regex 'entity' = 'slc0[^1-6].*' | distinct entity`	slc07cuc slc07ptt
To find the records by specifying the character class for the presence of the characters from3 to 8	`* | regex 'entity' = 'slc0[3-8].*' |distinct entity`	slc07cuc slc07ptt
To find the records by specifying the character class for the presence of the characters 1 or 2	`* | regex 'entity' = 's.*[12].*'|distinct entity`	slc07ptt:8452 stuyb43:1831

Rename Command

Use this command to change the name of a field.

After the name of a field has been changed through the Search bar, you can view the original field name and the changed field name from the UI.

Syntax

```
|rename <field> as <new_field>
```

Parameters

The following table lists the parameters used with this command, along with their descriptions.

Parameter	Description
`field`	Use this parameter to specify the original name of the field.
`new_field`	Use this parameter to specify the new name of the field.

For examples of using this command in typical scenarios, see:

- Histogram Chart Options in *Using Oracle Log Analytics*
- Analyze the Access Logs of Oracle WebLogic Server in *Using Oracle Log Analytics*
- Link by Using SQL Statement as the Field of Analysis in *Using Oracle Log Analytics*
- Analyze the Time Taken Between Steps in a Transaction in *Using Oracle Log Analytics*
- Use Link Navigation Functions to Identify Events in a Database in *Using Oracle Log Analytics*

The following query returns the count of logs from the different log sources, and renames the field `Host IP Address (Client)` and `clientIP`.

```
*|stats count by 'Log Source'|rename 'Host IP Address (Client)' as
clientip
```

Search Command

Use this command to search for a logical expression.

Syntax

```
search <logical_expression>
```

Where *`<logical_expression>`* includes the following:

- *`<keyword_expression>`*
- *`<comparison_expression>`*
- *`<cmp>`*
- *`<eval_expression>`*
- *`<value>`*
- *`<string_literal>`*
- *`<between_exp>`*
- *`<in_exp>`*

Parameters

The following table lists the parameters used with this query, along with their descriptions.

Parameter	Description
`logical_expression`	This parameter includes all keywords or fieldname-value pairs used to filter data.

Parameter	Description
keyword_expression	Use this parameter to specify the keywords or phrases you want to match. The value for this result must follow the format *<string>*, '*<string>*', or "*<string>*".
comparison_expression	Use this parameter to compare a field to an expression. The value for this parameter must follow the format *<field_name><cmp> <eval_expression>*, *<field_name> <between_exp>*, or *<field_name> <in_exp>*.
cmp	Use this parameter to specify a comparative operator. Permitted values for this parameter include =, !=, <, >, >=, <=, and [NOT] LIKE.
eval_expression	Use this parameter to specify literals which represent the value of your destination field.
value	Use this parameter to specify a numeric or a string literal. The permitted value for this parameter must follow the format *<string_literal>* or *<numeric literal>*.
string_literal	Use this parameter to specify a string literal. The permitted value for this parameter must follow the format "*<string>*", '*<string>*', or *<string>*.
between_exp	Use this parameter to specify a range. The permitted value for this parameter must follow the format [NOT] BETWEEN (*<value>* | *<numeric_literal>*) AND (*<value>* | *<numeric_literal>*).
in_exp	Permitted values for this parameter must follow the format [NOT] IN "("*<value>*) (","(*<value>*)*")".

For examples of using this command in typical scenarios, see:

- Search Logs Using Keywords and Phrases in *Using Oracle Log Analytics*

The following query returns ORA-00600 log entries.

```
Message like '%ORA-00600%'
```

The following query returns all ORA-00600 logs and fatal logs.

```
Message like 'ORA-600%' or Severity = fatal
```

The following query returns all database logs.

```
'Target Type' in ('Database Instance', 'Cluster Database')
```

The following query returns all logs for the database MyDb.

```
Target = MyDb and 'Target Type' = 'Database Instance'
```

SearchLookup Command

Use the `searchLookup` command to retrieve contents from a lookup table. By default all fields and contents are returned.

If you use the UI to run the `searchLookup` command, then ensure that the time range includes the time when the lookup was created. It is recommended that you create a Saved Search for using the `searchLookup` command.

Syntax

```
searchLookup  table=<lookupTable>
```

Parameters

The following table lists the parameters used in this command, along with their description.

Parameter	Description
`lookupTable`	Name of the lookup table.

The following example shows how to search for log records that contain the severity with value `info` in the table

```
searchlookup table = OraErrorCodes | search severity = info
```

The following example shows how to search for the fields in the table.

```
searchlookup table = OraErrorCodes | fields -*, errid, msg
```

Sort Command

Use this command to sort logs according to specified fields.

Syntax

```
sort [+|-] <field_name> (,[+|-]<field_name>)*
```

Parameters

The following table lists the parameters used with this command, along with their descriptions.

Parameter	Description
`field_name`	Use this parameter to specify the field according to which you want the results sorted.

Parameter	Description
`+ | -`	Specify + to sort the results in ascending order, and - to sort the results in descending order. If neither + nor - are specified, then the results are organized in ascending order, by default.

For examples of using this command in typical scenarios, see:

- Use Link Navigation Functions to Identify Events in a Database in *Using Oracle Log Analytics*

The following query returns the list of fatal logs, arranged in descending order, according to time.

```
Severity = fatal | fields Time, Target, 'Target Type' | sort -Time
```

Stats Command

Use this command to provide summary statistics, optionally grouped by a field. The output for this query includes one field for each of the fields specified in the query, along with one field for each aggregation.

Note:

While the `eval` command creates new fields by using existing fields and arbitrary expressions, the `stats` command calculates statistics based on existing fields.

Syntax

```
stats <stats_function> "("<field_name>")" [as new_field_name] [by
<field_name> (,<field_name>)*]
```

Parameters

The following table lists the parameters used with this command, along with their descriptions.

Parameter	Description
`field_name`	Use this parameter to specify the field according to which you want the results to be grouped.

Functions

The following table lists the functions available with this command, along with their examples.

Function	Examples
Values	`values(field_name)` • `values(Label)` • `values(Severity)` • `values('Client Host City')` • `values('Client Host Country')` • `values('Client Host Continent')`
Unique	`unique(field_name)` • `unique(Label)` • `unique(Severity)` • `unique('Client Host City')` • `unique('Client Host Country')` • `unique('Client Host Continent')`
Earliest	`earliest(field_name)` • `earliest('OS Process ID')`
Latest	`latest(field_name)` • `latest('Error ID')`
Trend	`trend(duration)` • `trend` • `trend(1hr)`
Average **Note**: This function is supported only for numeric fields.	`avg(field_name)` • `avg('Content Size')`
Distinct Count	`distinctcount(field_name)` • `distinctcount(Severity)`
Maximum **Note**: This function is supported only for numeric fields.	`max(field_name)` • `max('Content Size')`
Median **Note**: This function is supported only for numeric fields.	`median(field_name)` • `median('Content Size')`
Minimum **Note**: This function is supported only for numeric fields.	`min(field_name)` • `min('Content Size')`
n-th value **Note**: This function is supported only for numeric fields.	`pct(field_name, n)` • `pct('Content Size', 90)`
Sum **Note**: This function is supported only for numeric fields.	`sum(field_name)` • `sum('Content Size')`
Standard Deviation **Note**: This function is supported only for numeric fields.	`stddev(field_name)` • `stddev('Content Size')`

For examples of using this command in typical scenarios, see:

- Rename the Fields by Editing the Query in *Using Oracle Log Analytics*
- Histogram Chart Options in *Using Oracle Log Analytics*
- Visualize Time Series Data Using the Link Trend Feature in *Using Oracle Log Analytics*
- Analyze the Access Logs of Oracle WebLogic Server in *Using Oracle Log Analytics*

The following query returns the count of all logs grouped by severity, including those logs where the value of severity is null.

```
* | stats count by Severity
```

Running the following query excludes the results from the aggregation if a field value is null.

```
* | stats count(Severity) by Severity
```

The following query returns the count of fatal logs grouped by entity name and type.

```
Severity = fatal | stats count by Entity, 'Entity Type'
```

The following query returns the total count of logs.

```
* | stats count
```

The following query returns the count of database logs grouped by entity name and severity.

```
'Entity Type' = 'Database Instance' | stats count by Entity, Severity
```

The following query returns the values of severity grouped by entity name.

```
* | stats values(Severity) by Entity
```

The following query returns the unique values of client host city grouped by entity type.

```
* | stats unique('Client Host City') by 'Entity Type'
```

The following query returns the earliest values of the OS Process ID.

```
* | stats earliest('OS Process ID')
```

The following query returns the latest values of the Error ID.

```
* | stats latest('Error ID')
```

The following query creates an inlined timeseries sparkline. The default function is count

```
* | stats trend(avg(duration), 2min) by Entity
```

The following query returns the standard deviation of the set of numbers of the specified field

```
* | stats stddev('Content Size')
```

Tail Command

Use the `tail` command to display the last *n* number of results.

Syntax

```
tail [limit=<limit>]
```

Parameters

The following table lists the parameters used in this command, along with their descriptions.

Parameter	Description
`limit`	Specify the number of entries you want to see. If no value is specified, then the default value of `10` is used.

The following command returns the last 20 results with the severity of fatal.

```
Severity = fatal | tail limit = 20
```

Timecluster Command

Use this command to group the time-series charts together based on how similar they are to one another.

Syntax

```
timecluster [<timecluster_options>] <stats_function> (<field_name>)
[as new_field_name] [, <stats_function> (<field_name>) [as
new_field_name]]* by <field_name> [, <field_name>]*
```

Parameters

The following table lists the parameters used with this command, along with their descriptions.

Parameter	Description
`span`	Use this parameter to set the size of each bucket, using a span length based on time. Permitted values for this parameter must follow the format *`<int><timescale>`*.

Parameter	Description
`timescale`	Use this parameter to specify the time for sizing the buckets. Permitted values for this parameter must be either *<sec>*, *<min>*, *<hour>*, *<day>*, *<week>*, or *<mon>*. Syntax for the permitted values: • *sec*: `s` | `sec` | `secs` | `second` | `seconds` • *min*: `m` | `min` | `mins` | `minute` | `minutes` • *hour*: `h` | `hr` | `hrs` | `hour` | `hours` • *day*: `d` | `day` | `days` • *week*: `w` | `week` | `weeks` • *month*: `mon` | `month` | `months`
`field`	The field must have a timestamp value. If not, `Start Time` is used.
`limit`	Use this parameter to specify the number of values to return for each function.
`chart_name`	Use this parameter to specify the name to display for the chart.

Cluster time-series pattern by entity:

```
* | link Entity | stats sum('Content Size') as 'Content Size'  | timecluster
avg('Content Size') by Entity
```

Timestats Command

Use this command to generate data for displaying statistical trends over time, optionally grouped by field.

Syntax

```
timestats [<bucketing_option>] <stats_function / timestats_function>
"("<field_name>")" [as new_field_name] [by_<field_name>]
```

Parameters

The following table lists the parameters used with this command, along with their descriptions.

Parameter	Description
`bucketing_option`	Use this parameter to specify how the data should be bucketed. Permitted values for this parameter must follow the format *<span>*.
`span`	Use this parameter to set the size of each bucket, using a span length based on time. Permitted values for this parameter must follow the format *<int><timescale>*.
`timescale`	Use this parameter to specify the time for sizing the buckets. Permitted values for this parameter must be either *<sec>*, *<min>*, *<hour>*, *<day>*, *<week>*, *<month>*, or *<year>*.

Parameter	Description
`sec`	Use this parameter to specify whether the buckets should span across seconds. Permitted values for this parameter include `s`, `sec`, `secs`, `second`, and `seconds`.
`min`	Use this parameter to specify whether the bucket should span across minutes. Permitted values for this parameter include `m`, `min`, `mins`, `minute`, or `minutes`.
`hour`	Use this parameter to specify whether the bucket should span across hours. Permitted values for this parameter include `h`, `hr`, `hrs`, `hour`, and `hours`.
`week`	Use this parameter to specify whether the bucket should span across weeks. Permitted values for this parameter include `w`, `week`, and `weeks`.
`month`	Use this parameter to specify whether the bucket should span across months. Permitted values for this parameter include `mon`, `month`, and `months`.

Note:

You can use the functions that are associated with the `stats` command with the `timestats` command too. For details about the functions and the examples of using the functions with the command, see Stats Command.

Functions

The following table lists the functions available with this command, along with their examples.

Function	Examples
persecond: Returns one data point per span interval representing the average rate per second.	`persecond(field_name)` • `| timestats persecond('Error Id')`
perminute: Returns one data point per span interval representing the average rate per minute	`perminute(field_name)` • `| timestats perminute('Error Id')`
perhour: Returns one data point per span interval representing the average rate per hour	`perhour(field_name)` • `| timestats perhour('Error Id')`
perday: Returns one data point per span interval representing the average rate per day	`perday(field_name)` • `| timestats perday('Error Id')`

The following query returns the count of fatal log entries over the specified time range.

```
Severity = fatal | timestats count
```

The following query returns the count of logs bucketed into daily chunks.

```
* | timestats span = 1day count
```

Top Command

Use this command to display a specified number of results with the highest aggregated value as determined by the specified field. Since the field must represent an aggregated value, this command must be preceded by a `stats` or `cluster` command. The results from the command to the left of the pipe character are sorted in descending order, based on the field specified, and the requested number of results are displayed.

Syntax

```
top [limit=<limit>] <field_name>
```

Parameters

The following table lists the parameters used with this command, along with their descriptions.

Parameter	Description
`field_name`	Use this parameter to specify the field according to which the highest aggregated values are determined.
`limit`	Use this parameter to specify the limit. If no value is specified, then the default value of `10` is used. Entering a value of `-1` will return all rows.

The following query returns the 10 log sources with the highest number of log entries.

```
* | stats count as cnt by 'Log Source' | top cnt
```

The following query returns the 5 host entities with the most fatal log entries.

```
'Entity Type' = Host and Severity = fatal | stats count as cnt by Entity,
'Entity Type' | top limit = 5 cnt
```

The following query returns the 10 summaries with the highest number of similar log records.

```
* | cluster | top Count
```

Where Command

Use the `where` command to calculate the value of an expression to be true or false.

Syntax

```
*|where <expression>
```

Operators and Functions Available with the Command

The following table lists the operators available with the `where` command.

Category	Example
Arithmetic Operators	`+`, `-`, `*`, `/`, `%`
Comparison Operators	`=`, `!=`, `<`, `>`, `<=`, `>=`
Logical Operators	`and`, `or`, `not`
Conditional Operators	`if(<expression>,<expression>,<expression>)`
Multiple Comparison Operators	`in`, `not in`

The following table lists the functions available with the `where` command.

Category	Example
String Functions	• `concat(String, String)` • `indexof (String, String [,int])` • `length(String)` • `literal(String)` • `lower(String)` • `ltrim(String, Character)` • `replace(String, String, String)` • `rtrim(String, Character)` • `substr(String, int [, int])` • `todate(String [, format])` • `toduration(String)` • `tonumber(String)` • `trim(String)` • `trim(String, Character)` • `upper(String)` • `urldecode(String)`
Numeric Functions	• `abs(number)` • `ceil9number)` • `floor(number)` • `formatduration(number)` • `max(number, number)` • `min(number, number)` • `power(number, int)` • `round(number, int)` • `sqrt(number)` • `tostring(number)`
Date Functions	• `dateadd(date, property, amount)` • `dateset(date, property, value [, property, value])` • `formatdate(ate [,format])` • `now()`
Network Functions	`cidrmatch(String, String)`

Note:

- For the `concat()` function, you can input numeric data types like integer, float, or long. The numeric fields with be automatically converted to the corresponding string values.
- You can use `||` to concatenate *n* number of inputs. Here too, you can input numeric data types which will be automatically converted to the corresponding string values.

Parameters

The following table lists the parameters used in this command, along with their descriptions.

Parameter	Description
`boolean_expression`	Specify the expression for which the true or false value needs to be calculated.

For examples of using this command in typical scenarios, see:

- Link Visualization in *Using Oracle Log Analytics*
- Use Dictionary Lookup in Cluster in *Using Oracle Log Analytics*
- Use Dictionary Lookup in Link in *Using Oracle Log Analytics*
- Generate Link Alerts in *Using Oracle Log Analytics*
- Link by Cluster in *Using Oracle Log Analytics*

Following are some examples of the `eval` command.

```
*|where severity = FATAL
```

```
*|where 'Client Host City' = 'redwood city'
```

```
*|where upper(severity) = FATAL
```

```
*|where length(URI) >= 40
```

```
*|where replace('aabbcc', 'bb', 'xx') = aaxxcc
```

```
*|where concat(host, concat(':', port)) != hostname
```

```
*|where host || ':' || port != hostname
```

```
*|where substr('aabbcc', 2, 4) = bb
```

```
*|where round('Content Size') = 1000
```

```
*|where floor('Content Size') > 1000
```

```
*|where max('Content Size In', ''Content Size Out') < 1000
```

```
*|where
urldecode('http%3A%2F%2Fexample.com%3A893%2Fsolr%2FCORE_0_0%2Fquery')
= URI
```

```
*|where 'User Name' in (host1, host2) = omcuser
```

The following example compares the IP addresses in the field `srvrhostip` to a subnet range.

```
*|where cidrmatch(srvrhostip, '192.0.2.254/25')
```

The following example returns the string value of the field Delay.

```
*|where Status = literal(Delay)
```

The following example removes the matching character from both the ends.

```
*|where trim(Command,"\") = initparams
```

The following example removes the matching character from the left end.

```
*|where ltrim('Error ID',0) = 76890
```

The following example removes the matching character from the right end.

```
*|where rtrim('OS Process ID',2) = 3123
```

The following example compares the string `Start Time` with `1/1/18` in the date format `MM/dd/yy`.

```
*|where 'Start Time' > toDate('1/1/18', 'MM/dd/yy')
```

The following example calculates the difference between the values of `End Time` and `Start Time` and compares the string with the duration of `0:0:45`.

```
*|where 'End Time' - 'Start Time' > toDuration('0:0:45')
```

The following example specifies the format of the duration as `0:0:45.000`.

```
*|where formatDuration('End Time' - 'Start Time') = '0:0:45.000'
```

The following examples illustrate the use of date functions.

```
*|where 'Start Time' > dateAdd(now(), hour, -1)
*|where 'Start Time' > dateSet(now(), hour, 0, minute, 0, sec, 0, msec, 0)
*|where formatDate('Start Time', 'MM/dd/yyyy') = '01/15/2018'
*|where 'Start Time' - now() > 45000
```

The following example calculates the position of `.com` in the `uri` string and evaluates if it is not equal to `-1`.

```
*| where indexOf(uri, '.com') != -1
```

You can use the md5, sha1, and sha256 hash functions with the where command to filter log data. The following example evaluates if the value of the field `user` is `md5("jack")`.

```
*|where user = md5("jack")
```

B

Log Queries: Quick Reference

Here are some examples about how to phrase Search queries.

Search queries can be grouped as

- Reporting Queries
- Grouping Queries
- Filtering Queries

Reporting Queries

Requirement	Query
Return count of logs grouped by entity type, severity.	`* | stats count by 'entity type', severity`
Return time series for count of fatal logs.	`severity='fatal' | timestats count`
Return the top 5 entities and their type with fatal logs.	`severity='fatal' | stats count as 'fatal count' by entity, 'entity type' | top limit=5 'fatal count'`
Return the top 50 distinct entities as well as the count of logs for each of those entities.	`* | stats count as 'count by entity' by entity | top limit=50 'count by entity'`

Grouping Queries

Requirement	Query
Perform cluster analysis on fatal logs and save it in a collection called “Fatal logs”.	`severity='fatal' | cluster collection='fatal logs'`
Return the fatal logs that were included in summary ID 10002000002 and 10032000002 in the collection ‘Fatal logs’.	`Severity = fatal | clusterdetails collection = 'Fatal Logs' id in (10002000002, 10032000002)`

Filtering Queries

Requirement	Query
Return logs that do not contain 404 in their raw text.	`not 404`
Return logs that contain FAIL in their raw text or have a fatal severity.	`FAIL or Severity = fatal`