ALIBABA CLOUD

Alibaba Cloud

Log Service

Index and query

Document Version: 20220510

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Log analysis overview

Log Service provides the log analysis feature. This feature works together with the log search feature and is implemented by using the SQL syntax.

Syntax

Each query statement consists of a search statement and an analytic statement. The search statement and the analytic statement are separated by a vertical bar (|). A search statement can be executed alone. However, an analytic statement must be executed together with a search statement. You can use the log analysis feature to analyze data that meets specified search conditions in a Logstore. You can also use the feature to analyze all data in a Logstore.

> **Note** Analytic statements are not case-sensitive.

- Syntax

```
Search statement|Analytic statement
```

Statement	Description
Search statement	A search statement specifies one or more search conditions, and then returns the logs that meet the specified conditions. A search statement can be a keyword, a numeric value, a numeric value range, a space character, or an asterisk (*). If you specify a space character or an asterisk (*) as the search statement, no conditions are specified and all logs are returned. For more information, see Search syntax.
Analytic statement	An analytic statement is used to aggregate or analyze all log data or the log data that meets the specified search conditions in a Logstore.

- Example

```
* | SELECT status, count(*) AS PV GROUP BY status
```

Syntax description

Log Service allows you to analyze logs by using the standard SQL-92 syntax. When you use an analytic statement in Log Service, take note of the following points:

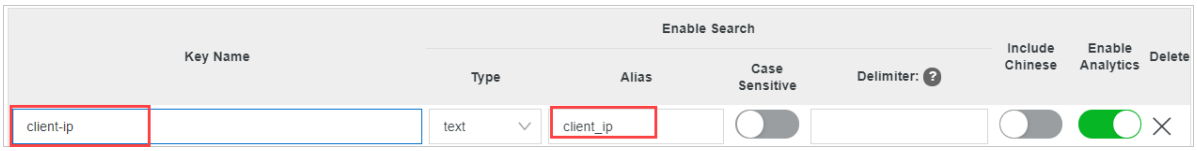
- You do not need to add a semicolon (;) at the end of the analytic statement to specify the end of the statement.
- If you do not want to use an SQL nested subquery, you do not need to specify a FROM or WHERE clause in the analytic statement. By default, all log data of the current Logstore is analyzed.
- You can use an SQL nested subquery to perform complex data analysis. If you want to use an SQL nested subquery, you must specify a FROM clause.

```
* | SELECT sum(pv) FROM (SELECT count(*) AS pv FROM log GROUP BY method)
```

- If you want to use a string in an analytic statement, you must enclose the string in single quotation marks (''). Strings that are not enclosed or enclosed in double quotation marks ("") indicate field names or column names. For example, 'status' indicates the status string, and status or "status" indicates the status log field.
- A column name that you specify in the analytic statement can contain only letters, digits, and underscores (_). The column name must start with a letter.

If you specify a column name that does not comply with the SQL-92 syntax when you collect logs, you must specify an alias for the column name when you configure indexes. For more information about how to configure indexes, see Configure indexes.

Notice Aliases are used only for SQL analysis. The original column names are used for storage. You must use the original column names in search statements and use aliases in analytic statements.



Limits

Item	Standard SQL	Dedicated SQL
Number of concurrent analytic statements	Each project supports a maximum of 15 concurrent analytic statements. For example, 15 users can concurrently execute analytic statements in all Logstores of a project.	Each project supports a maximum of 150 concurrent analytic statements. For example, 150 users can concurrently execute analytic statements in all Logstores of a project.
Data volume	Each shard supports only 1 GB of data for a single analytic statement.	An analytic statement can scan a maximum of 200 billion rows of data at the same time.
Method to enable	By default, Standard SQL is enabled.	A switch is provided for you to manually enable Dedicated SQL. For more information, see Enable Dedicated SQL.
Resource usage fee	Free of charge.	You are charged based on the actual CPU time. For more information, see Billable items.
Applicable scope	You can analyze only the data that is written to Log Service after the log analysis feature is enabled. If you want to analyze historical data, you must reindex the historical data. For more information, see Reindex logs for a Logstore.	You can analyze only the data that is written to Log Service after the log analysis feature is enabled. If you want to analyze historical data, you must reindex the historical data. For more information, see Reindex logs for a Logstore.
Returned result	By default, an analytic statement returns a maximum of 100 rows of data. If you want to view more data, use a LIMIT clause. For more information, see LIMIT clause.	By default, an analytic statement returns a maximum of 100 rows of data. If you want to view more data, use a LIMIT clause. For more information, see LIMIT clause.

Item	Standard SQL	Dedicated SQL
Size of a field value	The log analysis feature can analyze a maximum of 16,384 bytes (16 KB) of data in the value of each field. If the size of a field value exceeds 16 KB, the excess content is not analyzed. You can change the maximum size for each field value when you configure indexes. Valid values: 64 to 16384. Unit: bytes. For more information, see Configure indexes.	The log analysis feature can analyze a maximum of 16,384 bytes (16 KB) of data in the value of each field. If the size of a field value exceeds 16 KB, the excess content is not analyzed. You can change the maximum size for each field value when you configure indexes. Valid values: 64 to 16384. Unit: bytes. For more information, see Configure indexes.
Timeout period	The maximum timeout period for a single analytic statement is 55 seconds.	The maximum timeout period for a single analytic statement is 55 seconds.
Number of decimal places in the value of a double-type field	The value of a double-type field can contain a maximum of 52 decimal places. If the number of decimal places is greater than 52, the accuracy of the field value is compromised.	The value of a double-type field can contain a maximum of 52 decimal places. If the number of decimal places is greater than 52, the accuracy of the field value is compromised.

Analysis methods

> **Notice**
> - If you want to use the log analysis feature, you must turn on **Enable Analytics** for the required fields when you configure indexes. For more information, see Configure indexes.
> - Log Service provides reserved fields. For more information about how to analyze reserved fields, see Reserved fields.

- Use the Log Service console

  Log on to the Log Service console. On the Search & Analysis page of a Logstore, specify a time range and execute a query statement. For more information, see Query and analyze logs.

- Call the API or use an SDK

  Call the GetLogs or GetHistograms operation to query and analyze logs.

Analytic functions and syntax

This section lists the analytic functions and syntax that are supported by Log Service.

- SQL functions
  - Aggregate function
  - String functions
  - Date and time functions
  - JSON functions
  - Regular expression functions
  - Interval-valued comparison and periodicity-valued comparison functions
  - Array functions and operators
  - Map functions and operators
  - Mathematical calculation functions
  - Mathematical statistics functions

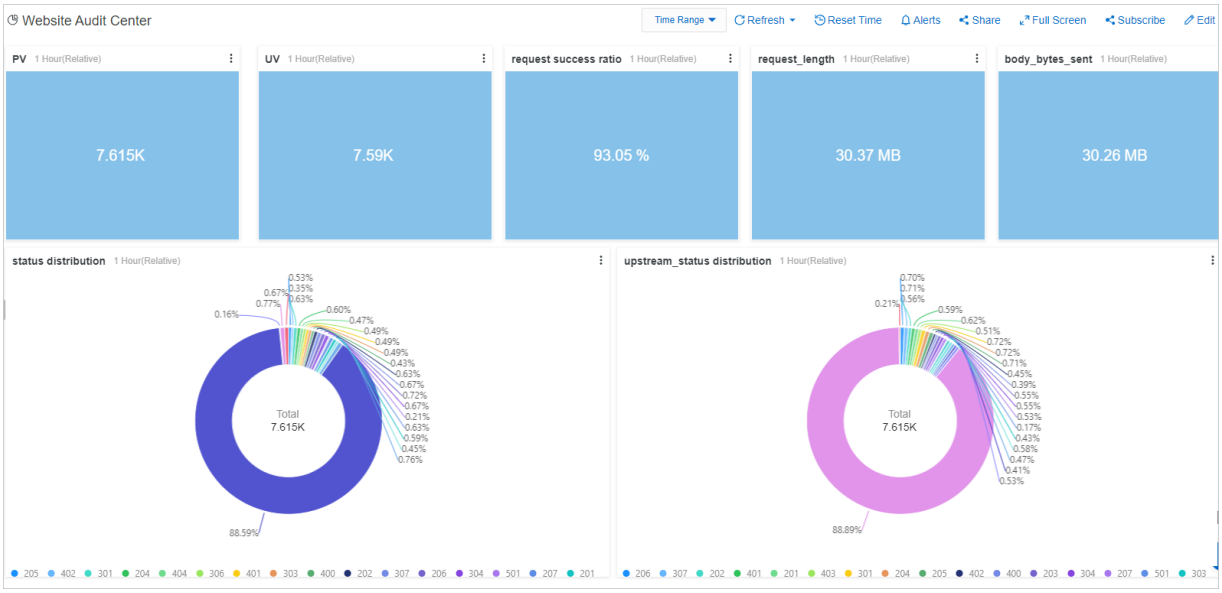
- Data type conversion functions
- Security check functions
- Window functions
- IP functions
- URL functions
- Approximate functions
- Binary functions
- Bitwise functions
- Geospatial functions
- Geo functions
- Color functions
- HyperLogLog functions
- Comparison operators
- Logical operators
- Unit conversion functions
- Window funnel function
- Lambda expressions
- Conditional expressions

- Machine learning syntax and functions
  - Smooth functions
  - Multi-period estimation functions
  - Change point detection function
  - Maximum value detection functions
  - Prediction and anomaly detection functions
  - Sequence decomposition function
  - Time series clustering functions
  - Frequent pattern statistical function
  - Differential pattern statistical function
  - Request URL classification function
  - Root cause analysis function
  - Correlation analysis functions
  - Kernal density estimation functions
  - Time series padding function
  - Anomaly comparison function
- SQL syntax
  - EXCEPT clause
  - EXISTS clause
  - GROUP BY clause
  - HAVING clause
  - INSERT INTO clause
  - INTERSECT clause
  - JOIN clause
  - LIMIT clause
  - ORDER BY clause

- UNION clause
- UNNEST clause
- VALUES clause
- WITH clause

Sample analysis results

The following figure shows a sample dashboard that displays the analysis results. For more information, see Visualization overview.



2.Reserved fields

When you collect logs or ship logs to other cloud services, Log Service adds log sources and timestamps to the logs in the form of key-value pairs. These fields are reserved in Log Service. This topic describes the reserved fields of Log Service.

Notice

- When you add Logtail configurations or call API operations to write log data, you cannot set the names of fields (keys) to be the same as those of reserved fields. If the names are the same, inaccurate queries or other issues may occur due to duplicate field names.
- Log Service does not ship fields that are prefixed by `__tag__`.
- You are charged for the fields that you add for logs based on the pay-as-you-go billing method. If you enable the indexing feature for the fields, you are also charged a small fee for index traffic and storage. For more information, see Pay-as-you-go.

Field	Type	Index and log analysis settings	Description
`__time__`	Integer (UNIX timestamp)	• Index settings: The `__time__` field is specified by using the from and to parameters in API operations. You do not need to create an index for this field. • Log analysis settings: If you turn on the Enable Analytics switch for a column, the log analysis feature is enabled for the `__time__` field by default.	The time when log data is written to a Logstore. This field can be used to ship, query, and analyze logs.
`__source__`	String	• Index settings: If you enable the indexing feature, Log Service creates an index for the `__source__` field by default. The index is of the text type. No delimiter is specified for the indexes. To query logs based on the index on this field, you can enter `source:127.0.0.1` or `__source__:127.0.0.1`. • Log analysis settings: If you turn on the Enable Analytics switch for a column, the log analysis feature is enabled for the `__source__` field by default.	The machine from which logs are collected. This field can be used to ship, query, analyze, and consume logs.

Field	Type	Index and log analysis settings	Description
`__topic__`	String	• Index settings: If you enable the indexing feature, Log Service creates an index for the `__topic__` field by default. The index is of the text type. No delimiter is specified for the indexes. To query logs based on the index of this field, you can enter `__topic__:XXX`. • Log analysis settings: If you turn on the Enable Analytics switch for a column, the log analysis feature is enabled for the `__topic__` field by default.	The topic of a log. If you specify a topic for a log, Log Service adds a topic field to the log. The key of the field is `__topic__` and the value of the field is the topic content. This field can be used to ship, query, analyze, and consume logs. For more information, see Log topic.
`_extract_others_`	String, which can be deserialized into a JSON map	This field does not exist in logs. You do not need to create an index for this field.	This field is the same as the `__extract_others__` field. We recommend that you use the `__extract_others__` field.
`__tag__:__client_ip__`	String	• Index settings: If you enable the indexing feature, Log Service creates indexes for all fields by default. The indexes are of the text type. No delimiter is specified for the indexes. Exact match and fuzzy match are supported for log queries that are based on the index field. • Log analysis settings: By default, the log analysis feature is disabled for the column indicated by this field. To enable the log analysis feature for this field, you must create an index for the `__tag__:__client_ip__` field and turn on the Enable Analytics switch.	The public IP address of the machine from which logs are collected. This field is a system tag. If you enable the Log Public IP feature, Log Service adds this field to each raw log that is collected from the log source. This field can be used to query, analyze, and consume logs. When you execute SQL statements to analyze this field, you must enclose this field in double quotation marks (""). For more information, see Tags and Log Public IP.

Field	Type	Index and log analysis settings	Description
`__tag__:__receive_time__`	String, which can be converted to an integer in the UNIX timestamp format	• Index settings: If you enable the indexing feature, Log Service creates indexes for all tags by default. The indexes are of the text type. No delimiter is specified for the indexes. Exact match and fuzzy match are supported for log queries that are based on the index field. • Log analysis settings: By default, the log analysis feature is disabled for the column indicated by this field. To enable the log analysis feature for this field, you must create an index for the `__tag__:__receive_time__` field and turn on the Enable Analytics switch.	The time when Log Service receives a log. This field is a system tag. If you enable the Log Public IP feature, Log Service adds this field to each raw log that is collected from the log source. This field can be used to query, analyze, and consume logs. For more information, see Tags and Log Public IP.
`__tag__:__path__`	String	• Index settings: If you enable the indexing feature, Log Service creates an index for the `__tag__:__path__` field by default. The index is of the text type. No delimiter is specified for the index. To query logs based on the index of this field, you can enter `__tag__:__path__:XXX`. • Log analysis settings: By default, the log analysis feature is disabled for the column indicated by this field. To enable the log analysis feature for this field, you must create an index for the `__tag__:__path__` field and turn on the Enable Analytics switch.	The path to the log files collected by Logtail. Logtail automatically adds this field to logs. This field can be used to query, analyze, and consume logs. When you execute SQL statements to analyze this field, you must enclose this field in double quotation marks ("").
`__tag__:__hostname__`	String	• Index settings: If you enable the indexing feature, Log Service creates an index for the `__tag__:__hostname__` field by default. The index is of the text type. No delimiter is specified for the index. To query logs based on the index of this field, you can enter `__tag__:__hostname__:XXX`. • Log analysis settings: By default, the log analysis feature is disabled for the column indicated by this field. To enable statistics for this field, create an index on the `__tag__:__hostname__` field and enable the log analysis feature.	The hostname of the machine from which Logtail collects logs. Logtail automatically adds this field to logs. This field can be used to query, analyze, and consume logs. When you execute SQL statements to analyze this field, you must enclose this field in double quotation marks ("").

Field	Type	Index and log analysis settings	Description
`__raw_log__`	String	You must create and configure an index of the text type for this field and enable the log analysis feature based on your business requirements.	The raw logs that fail to be parsed. If you disable the Drop Failed to Parse Logs feature, Logtail uploads raw logs that fail to be parsed. The key of this field is `__raw_log__` and the value of this field is the log content. This field can be used to ship, query, analyze, and consume logs. For more information, see Drop Failed to Parse Logs.
`__raw__`	String	You must create and configure an index of the text type for this field and enable the log analysis feature based on your business requirements.	The raw logs that are parsed. If you enable the Upload Raw Log feature, Logtail uploads the raw logs in the `__raw__` field together with the parsed logs. This field can be used in log audit and compliance check scenarios. This field can be used to ship, query, analyze, and consume logs. For more information, see Upload raw log.

3.Data types

When you configure indexes, you can set the data type of a field to text, long, double, or JSON. This topic describes the data types of fields and provides some examples.

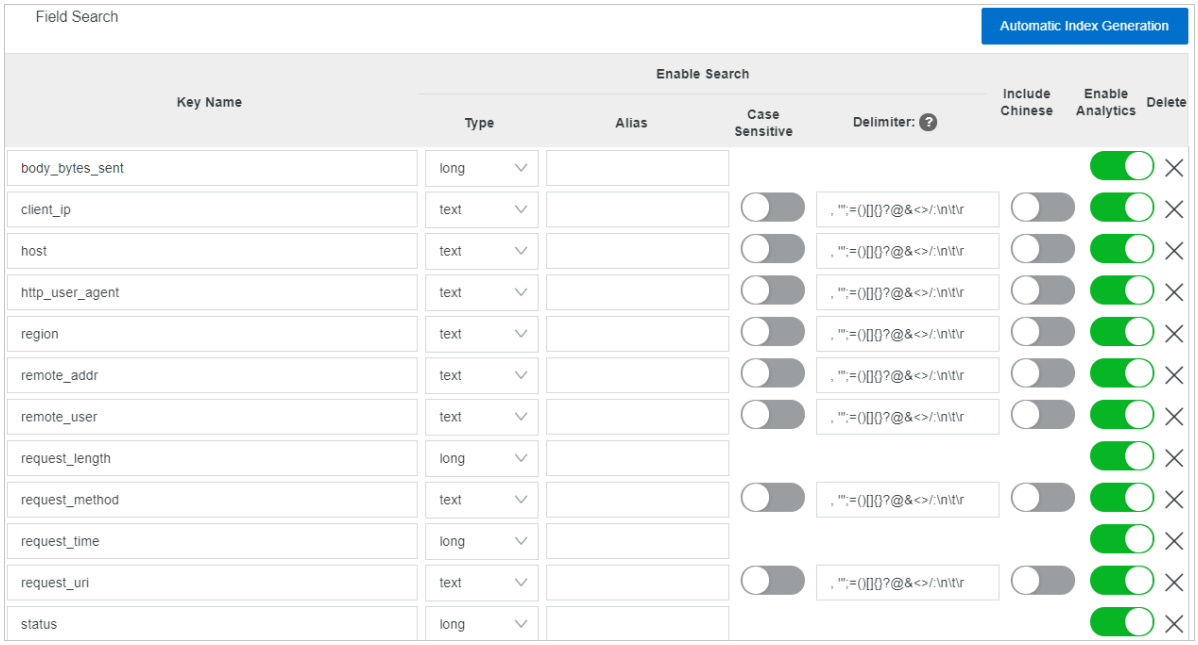
Text type

To query and analyze log data of the string type, you must set the data type of the related fields to text when you configure indexes. You must also turn on the Enable Analytics switch for these fields.

> **Note** By default, after you turn on the Full Text Index switch, the data type of all fields except the __time__ field is set to text.

- Sample log entry

- Index configurations



- Query statements
  - To query the log entries that do not contain GET requests, execute the following search statement:

```
not request_method : GET
```

- To query the log entries that start with cn, execute the following search statement:

```
cn*
```

- To collect the statistics on the distribution of clients, execute the following query statement:

```
* | SELECT ip_to_province(client_ip) as province, count(*) AS pv GROUP BY province ORDER BY pv
```

Long and double types

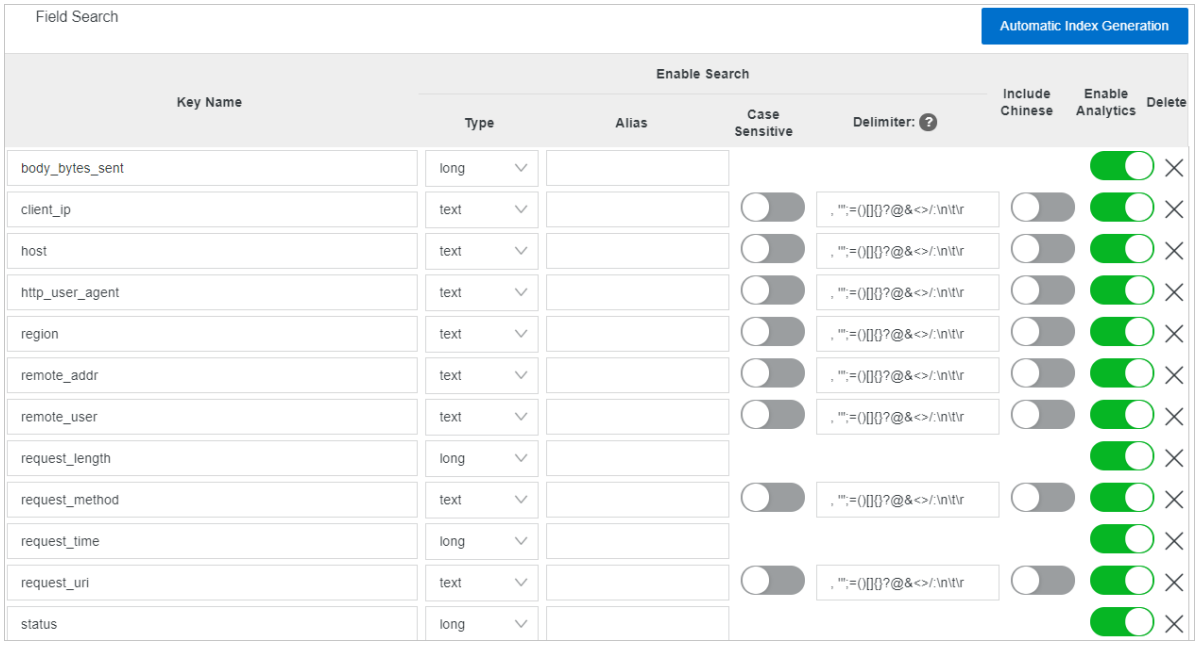
You can query the value of a field by using a numeric range only after you set the data type of the field to long or double.

- If the value of a log field is an integer, we recommend that you set the data type of the field to long when you configure indexes.
- If the value of a log field is a floating-point number, we recommend that you set the data type of the field to double when you configure indexes.

> **Notice**
>
> - If you set the data type of a field to long and the actual field value is a floating-point number, the field cannot be queried.
> - If you set the data type of a field to long or double and the actual field value is a string, the field cannot be queried.
> - If you set the data type of a field to long or double, you cannot use asterisks (*) or question marks (?) to perform fuzzy searches.
> - If the value of a field is an invalid numeric value, you can query data by using the **not key > -1000000** search statement. This not key > -1000000 search statement returns the log entries whose value of the field is an invalid numeric value. -1000000 can be replaced by a valid value that is smaller or equal to the smallest valid value of the field in your log entries.

- Sample log entry

- Index configurations



- Query statements
  - To query the log entries whose request duration is greater than 60 seconds, execute the following search statement:

    ```
    request_time > 60
    ```

  - To query the log entries whose request duration is greater than or equal to 60 seconds and less than 200 seconds, execute one of the following search statements:

    ```
    request_time in [60 200)
    ```

    ```
    request_time >= 60 and request_time < 200
    ```

  - To query the log entries whose response status code is 200, execute the following search statement:

    ```
    status = 200
    ```

JSON type

If the value of a field is in the JSON format, you can set the data type of the field to JSON when you configure indexes.

- You can set the data type of a field in JSON objects to long, double, or text based on the field value, and turn on the Enable Analytics switch. After you turn on the switch, Log Service allows you to query and analyze fields in JSON objects.
- For partially valid JSON-formatted data, only the valid parts can be parsed in Log Service.

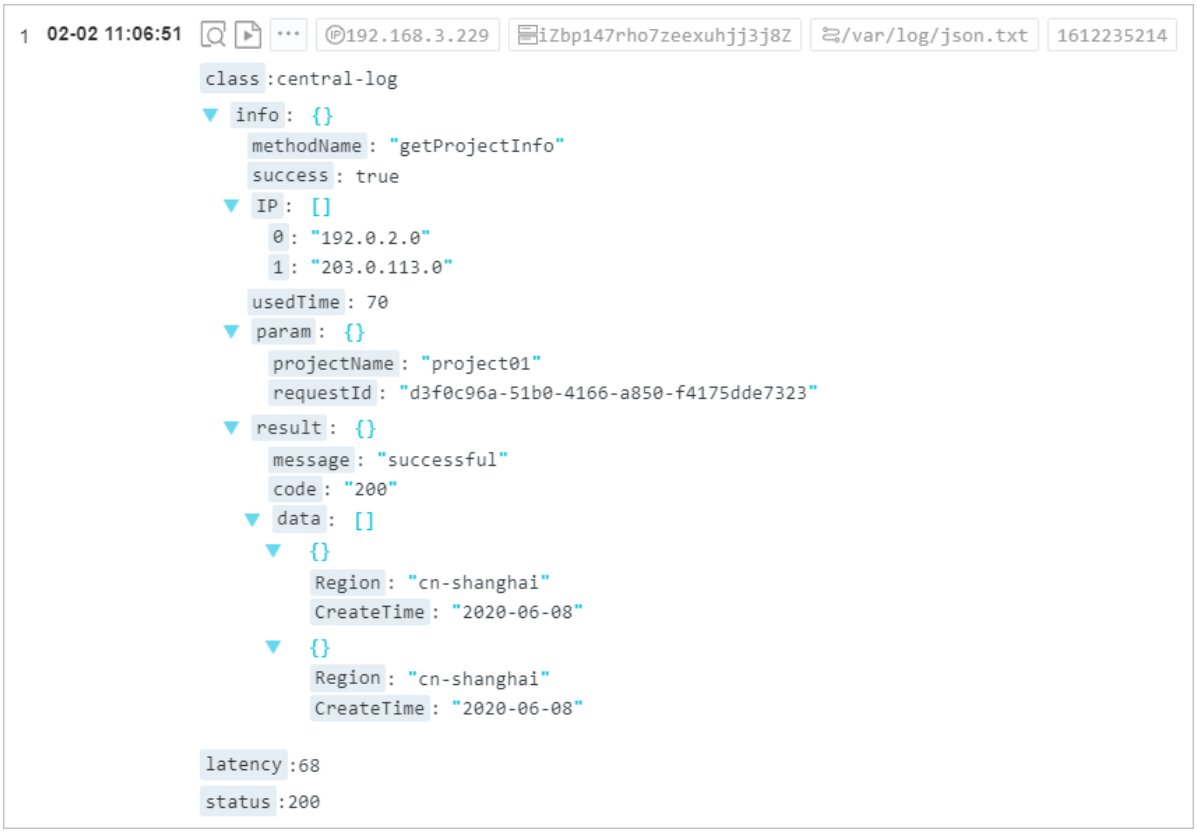
  The following example shows an incomplete JSON log entry. Log Service can parse the conctent.remote_addr, content.request.request_length, and content.request.request_method fields.

  ```
  content: {
       remote_addr:"192.0.2.0"
       request: {
              request_length:"73"
              request_method:"GE
  ```

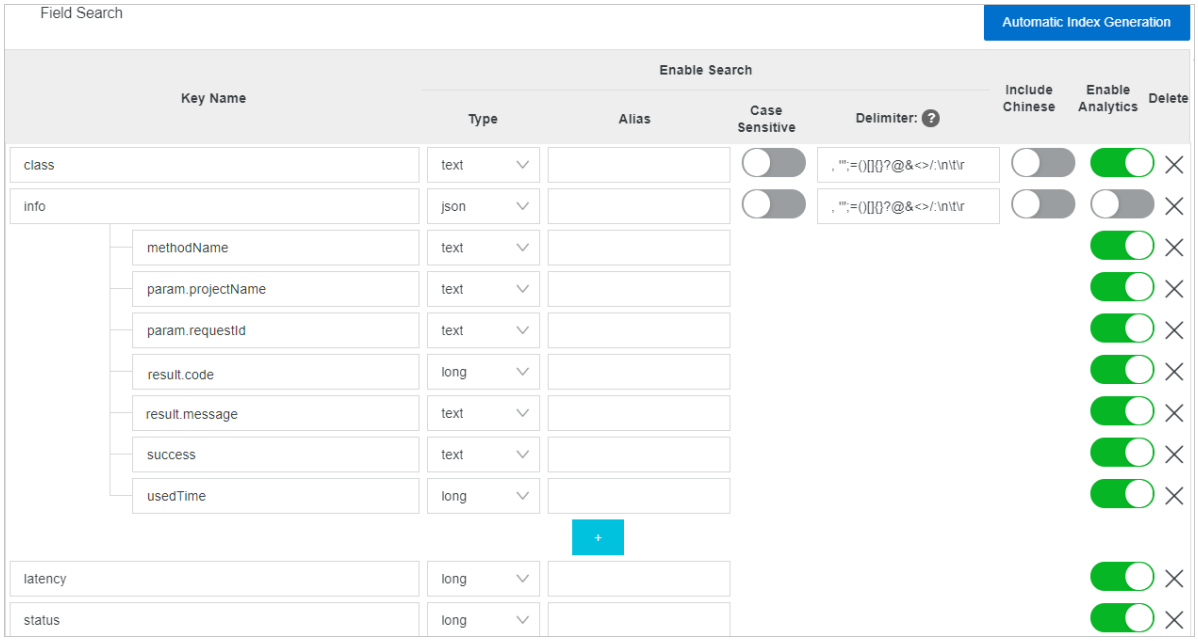
> **Notice**
>
> - Log Service allows you to configure indexes for leaf nodes in JSON objects. However, you cannot configure indexes for child nodes that contain leaf nodes.
> - You cannot configure indexes for fields whose values are JSON arrays. In addition, you cannot configure indexes for fields in a JSON array.
> - If the value of a field is of the Boolean type, you can set the data type of the field to text when you configure indexes.
> - The format of a query statement is `Search statement | Analytic statement`. In an analytic statement, you must enclose a field name by using double quotation marks ("") and enclose a string by using single quotation marks ('').

- Sample log entry

  The following figure provides an example of a JSON log entry. This log entry contains the reserved fields of Log Service. This log entry also contains the class, latency, status, and info fields. The value of the info field is a JSON object that contains multiple layers.



- Index configurations



In the preceding figure, you must take note of the following limits:

- The values of IP and data fields are JSON arrays. Therefore, you cannot configure indexes for the IP and data fields. You cannot query and analyze data by using these two fields.
- The region and CreateTime fields are in a JSON array. Therefore, you cannot configure indexes for the region and CreateTime fields. You cannot query and analyze data by using these two fields.

- Query statements
  - To query the log entries whose value of the usedTime field is greater than 60 seconds, execute the following search statement:

    ```
    info.usedTime > 60
    ```

  - To query the log entries whose value of the success field is true, execute the following search statement:

    ```
    info.success : true
    ```

  - To query the log entries whose usedTime field value is greater than 60 seconds and the projectName field value is not project01, execute the following search statement:

    ```
    info.usedTime > 60 not info.param.projectName : project01
    ```

  - To calculate the average duration that is used to obtain the details of the current project, execute the following query statement:

    ```
    methodName = getProjectInfo | SELECT avg("info.usedTime") AS avg_time
    ```

4.Configure indexes

Indexes are used in a storage structure to sort one or more columns of log data. You can query and analyze log data only after you configure indexes. Query and analysis results vary based on index configurations. Therefore, you must configure indexes based on your business requirements. If you configure both full-text indexes and field indexes, the configurations of the field indexes take precedence.

Prerequisites

Logs are collected. For more information, see Log collection methods.

> Notice
>
> - After you enable the indexing feature, you are charged for the index traffic and storage space occupied by indexes. For more information, see Billable items.
> - The indexing feature takes effect only on the log data that is written after you configure indexes. If you want to query and analyze historical data, you must use the reindexing feature. For more information, see Reindex logs for a Logstore.
> - By default, indexes are configured for specific reserved fields in Log Service. For more information, see Reserved fields. No delimiters are specified for the indexes of the `__topic__` and `__source__` fields. When you search for the fields, only exact match is supported.
> - If you want to search for the fields that are prefixed with __tag__, you must configure field indexes. Full-text indexes are not supported.

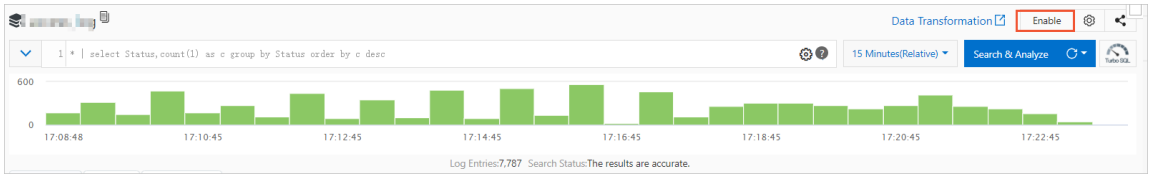
Index types

The following table describes the index types supported by Log Service.

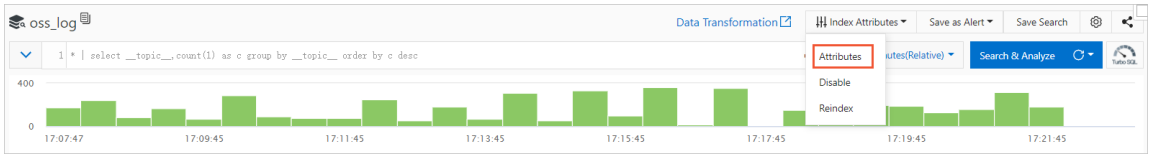
Index type	Description
Full-text index	Log Service splits an entire log into multiple words based on specified delimiters to create indexes. In a search statement, the field names (keys) and field values (values) are both plain text. For example, the search statement `error` returns the logs that contain the keyword `error`.
Field index	After you configure field indexes, you can specify field names and field values in the Key:Value format to search for logs. For example, the search statement `level:error` returns the logs whose `level` field value contains `error`. If you want to use the analysis feature, you must configure field indexes and turn on Enable Analytics for the required fields. If you turn on Enable Analytics, no additional index traffic is generated, and no additional storage space is occupied.

Configure full-text indexes

1.
2.
3.
4. Go to the index configuration page.
   - If you have not enabled the indexing feature, click **Enable** on the search and analysis page of the Logstore.



- If you have enabled the indexing feature, choose **Index Attributes > Attributes** on the search and analysis page of the Logstore.



5. In the **Search & Analysis** panel, configure the following parameters and click **OK**.

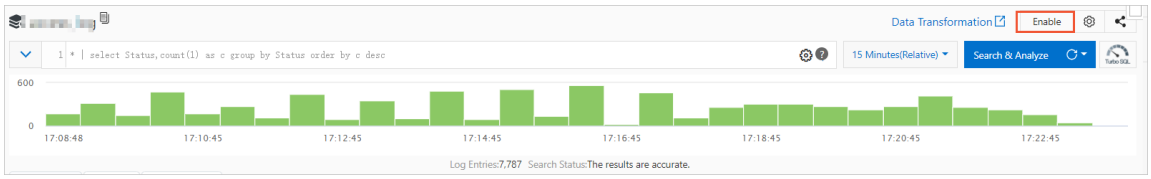
> **Note** The index configurations take effect within 1 minute.

Parameter	Description
LogReduce	If you turn on **LogReduce**, Log Service automatically aggregates text logs that have the same pattern during log collection. This way, you can obtain the overall information about logs. For more information, see LogReduce.
Full Text Index	If you turn on **Full Text Index**, the full-text indexing feature is enabled.
Case Sensitive	Specifies whether searches are case-sensitive. - If you turn on **Case Sensitive**, searches are case-sensitive. For example, if a log contains `internalError`, you can search for the log only by using the keyword `internalError`. - If you turn off **Case Sensitive**, searches are not case-sensitive. For example, if a log contains `internalError`, you can search for the log by using the keyword `INTERNALERROR` or `internalerror`.
Include Chinese	Specifies whether to distinguish between Chinese content and English content in searches. - After you turn on **Include Chinese**, if a log contains Chinese characters, the Chinese content is split based on the Chinese grammar. The English content is split based on specified delimiters. **Notice** When the Chinese content is split, the write speed is reduced. Proceed with caution. - If you turn off **Include Chinese**, all the content in a log is split based on specified delimiters.

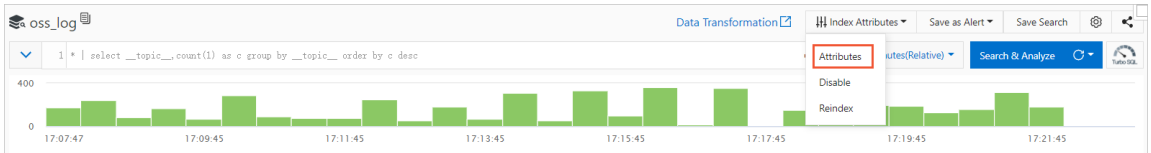
Parameter	Description
Delimiter	The delimiters that are used to split the content of a log into multiple words. Supported delimiters include `, '";=()[]{}?@&<>/:\n\t\r`. `\n` indicates a line feed, `\t` indicates a tab character, and `\r` indicates a carriage return. For example, the content of a log is `/url/pic/abc.gif`. ○ If you do not specify a delimiter, the log is regarded as a single word `/url/pic/abc.gif`. You can search for the log only by using the keyword `/url/pic/abc.gif` or by using `/url/pic/*` to perform a fuzzy search. ○ If you set the delimiter to a forward slash (/), the content of the log is split into the following three words: `url`, `pic`, and `abc.gif`. You can search for the log by using the keyword `url`, `abc.gif`, or `/url/pic/abc.gif`, or by using `pi*` to perform a fuzzy search. ○ If you set the delimiter to a forward slash (/) and a period (.), the content of the log is split into the following four words: `url`, `pic`, `abc`, and `gif`.
Maximum Statistics Field Length	The maximum length of a field value that can be retained for analysis. Default value: 2048. Unit: bytes. The default value is equal to 2 KB. You can change the value of the **Maximum Statistics Field Length** parameter. Valid values: 64 to 16384. **Notice** If the length of a field value exceeds the value of this parameter, the field value is truncated, and the excess part is not involved in analysis.

Configure field indexes

1.
2.
3.
4. Go to the index configuration page.
   - If you have not enabled the indexing feature, click **Enable** on the search and analysis page of the Logstore.



   - If you have enabled the indexing feature, choose **Index Attributes > Attributes** on the search and analysis page of the Logstore.



5. In the **Search & Analysis** panel, configure the following parameters and click **OK**.

Note The index configurations take effect within 1 minute.

Parameter	Description
Key Name	The name of the log field. Example: client_ip. **Note** ◦ If you configure an index for a tag field, you must set the **Key Name** parameter in the __tag__:KEY format. For example, you can set the parameter to __tag__:__receive_time__. Different tag fields are supported. For example, a tag field can indicate a public IP address or a UNIX timestamp. For more information, see Reserved fields. ◦ When you configure an index for a tag field, numeric data types are not supported. You must set the **Type** parameter for each tag field to **text**.
Type	The data type of the log field value. Valid values: text, long, double, and json. For more information, see Data types. **Note** If a field is of the long or double type, you cannot configure the **Case Sensitive**, **Include Chinese**, or **Delimiter** parameter.
Alias	The alias of the field. Example: ip. An alias is used only in analytic statements. You must use the original field name in search statements. For more information, see Column aliases.
Case Sensitive	Specifies whether searches are case-sensitive. ◦ If you turn on **Case Sensitive**, searches are case-sensitive. For example, if a log contains `internalError`, you can search for the log only by using the keyword `internalError`. ◦ If you turn off **Case Sensitive**, searches are not case-sensitive. For example, if a log contains `internalError`, you can search for the log by using the keyword `INTERNALERROR` or `internalerror`. ◦
Delimiter	The delimiters that are used to split the content of a log into multiple words. Supported delimiters include `, '";=()[]{}?@&<>/:\n\t\r`. `\n` indicates a line feed, `\t` indicates a tab character, and `\r` indicates a carriage return. For example, the content of a log is `/url/pic/abc.gif`. ◦ If you do not specify a delimiter, the log is regarded as a single word `/url/pic/abc.gif`. You can search for the log only by using the keyword `/url/pic/abc.gif` or by using `/url/pic/*` to perform a fuzzy search. ◦ If you set the delimiter to a forward slash (/), the content of the log is split into the following three words: `url`, `pic`, and `abc.gif`. You can search for the log by using the keyword `url`, `abc.gif`, or `/url/pic/abc.gif`, or by using `pi*` to perform a fuzzy search. ◦ If you set the delimiter to a forward slash (/) and a period (.), the content of the log is split into the following four words: `url`, `pic`, `abc`, and `gif`.

Parameter	Description
Include Chinese	Specifies whether to distinguish between Chinese content and English content in searches. ○ After you turn on **Include Chinese**, if a log contains Chinese characters, the Chinese content is split based on the Chinese grammar. The English content is split based on specified delimiters. **Notice** When the Chinese content is split, the write speed is reduced. Proceed with caution. ○ If you turn off **Include Chinese**, all the content in a log is split based on specified delimiters. ○
Enable Analytics	Before you can use the analysis feature, you must turn on **Enable Analytics**.
Maximum Statistics Field Length	The maximum length of a field value that can be retained for analysis. Default value: 2048. Unit: bytes. The default value is equal to 2 KB. You can change the value of the **Maximum Statistics Field Length** parameter. Valid values: 64 to 16384. **Notice** If the length of a field value exceeds the value of this parameter, the field value is truncated, and the excess part is not involved in analysis.

Index traffic descriptions

After you configure indexes, index traffic is generated.

Index type	Description
Full-text index	All field names and field values are stored as text. The field names and field values are both included in the index traffic.
Field index	The method that is used to calculate index traffic varies based on the data type of a field. • text: Field names and field values are both included in the index traffic. • long and double: Field names are not included in the index traffic. Each field value is counted 8 bytes in the index traffic. For example, if you configure an index of the long type for the status field and the field value is400, the string status is not included in the index traffic, and the 400 value is counted 8 bytes in the index traffic. • json: Field names and field values are both included in the index traffic. The subfields that are not indexed are also included. For more information, see How do I calculate index traffic for a JSON field? ○ If a subfield is not indexed, the index traffic is calculated by regarding the data type of the subfield as text. ○ If a subfield is indexed, the index traffic is calculated based on the data type of the subfield. The data type can be text, long, or double.

5.Query and analyze logs

After indexes are configured, you can query and analyze the logs that are collected on the query and analysis page in real time.

Prerequisites

- Logs are collected. For more information, see Data collection overview.
- Indexes are configured. For more information, see Configure indexes.

Procedure

1.
2.
3.
4. Enter a query statement in the input field.

   A query statement consists of a search statement and an analytic statement in the format of Search statement|Analytic statement. For more information, see Search syntax and SQL syntax.
5. Click **15 Minutes(Relative)** to specify the time range for the query statement.

   You can select a relative time, select a time frame, or customize a time range. However, the time range that you can specify is only accurate to the minute at most. If you want to use a time range that is accurate to the second, you must specify the time range in the analytic statement. Example: `* | SELECT * FROM log WHERE __time__>1558013658 AND __time__< 1558013660` .

   > **Note** The query and analysis results may contain logs that are generated 1 minute earlier or later than the specified time range.
6. Click **Search & Analyze** to view the query and analysis results.

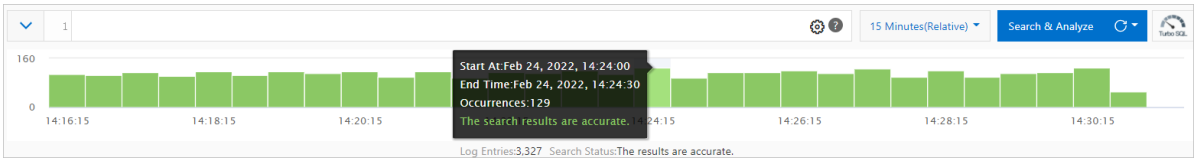
Manage query and analysis results

Log Service displays query and analysis results in a log distribution histogram, on the Raw Logs tab, and on the Graph tab. Log Service allows you to perform operations on the results. For example, you can configure alerts and create saved searches.

> **Note** When you execute a query statement, only 100 lines of data is returned by default. You can use a LIMIT clause to specify the number of lines that can be returned. For more information, see LIMIT clause.

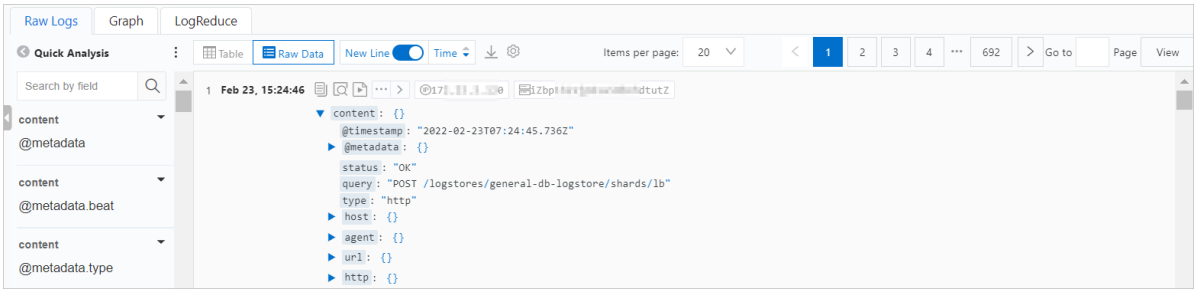
- Log distribution histogram

  The log distribution histogram shows the distribution of returned logs in different periods of time.

  - When you move the pointer over a green rectangle, you can view the period of time that is represented by the rectangle and the number of returned logs within the period.
  - If you click a green rectangle, you can view log distribution at a finer-grained level. In addition, you can view the returned logs within the period of time on the **Raw Logs** tab.
- Raw Logs tab

The **Raw Logs** tab displays the logs that are queried. You can click the **Table** or **Raw Data** tab to view the logs and perform the following operations:



- Quick Analysis: You can analyze the distribution of a field within a period of time. For more information, see Quick analysis.

  You can click the icon to specify whether to show the names or aliases of fields. You can create aliases when you configure indexes. For example, if the alias of host_name is host, host is displayed in the Quick Analysis list after you select Show Field Aliases.

  > **Note** If a field does not have an alias, the name of the field is displayed in the Quick Analysis list even if you select Show Field Aliases.

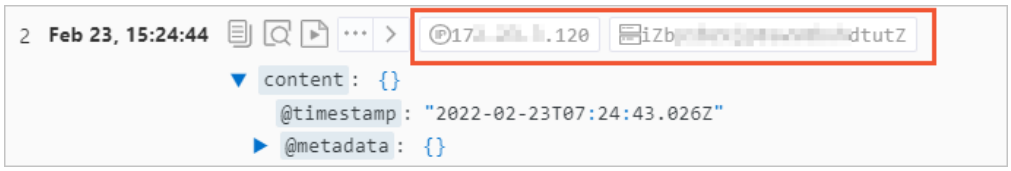
- Context query: On the **Raw Data** tab, you can find a log and click the icon to query the context information about the log in the raw log file. For more information, see Context query.

  > **Note** You can perform context query only on the logs that are collected by Logtail.

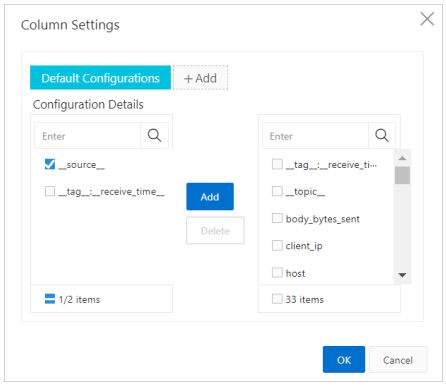
- LiveTail: On the **Raw Data** tab, you can find a log and click the icon to monitor logs in real time and extract important information from the logs. For more information, see LiveTail.

  > **Note** You can use LiveTail only on the logs that are collected by Logtail.

- Tag Configurations: On the **Raw Data** tab, you can click the icon and select **Tag Configurations** to hide less important fields.



- Column Settings: On the **Table** tab, you can click the icon and select **Column Settings** to specify the columns that you want to display in the table. The column names are field names, and the column content is field values.



- JSON Configurations: On the **Table** or **Raw Data** tab, you can click the icon and select **JSON Configurations** to specify the level for JSON expansion.
- Event Settings: On the **Table** or **Raw Data** tab, you can click the icon and select **Event Settings** to configure events for raw logs. For more information, see Configure events.
- Log Download: On the **Table** or **Raw Data** tab, you can click the icon to download logs. You can specify the tool that is used to download logs and the range of logs to download. For more information, see 下载日志.

- Graph tab

  After you execute a query statement, you can view the query and analysis results on the **Graph** tab.
  - View query and analysis results: Log Service renders the results of the query statement to charts. Log Service provides various types of charts, such as tables, line charts, and column charts. For more information, see Chart overview.
  - Add a chart to a dashboard: Log Service provides dashboards on which you can analyze data in real time. You can click **Add to New Dashboard** to save the query and analysis results as a chart to a dashboard. For more information, see Visualization overview.
  - Configure interactive events: Interactive events are important for data analysis. You can use interactive events to switch between the levels of data dimensions and the analysis granularities to obtain more detailed information. Interactive events include events to open a Logstore, open quick analysis, open a dashboard, open trace analysis, open trace details, and customize an HTTP link. For more information, see Configure a drill-down event for a chart.

- LogReduce tab

  On the **LogReduce** tab, you can click **Enable LogReduce** to cluster similar logs during log collection. For more information, see LogReduce.

- Alerting

  On the query and analysis page, you can choose **Save as Alert > New Alert** to configure alerts based on the query and analysis results. For more information, see Configure an alert in Log Service.

- Saved search

  On the query and analysis page, you can click **Save Search** to save a query statement as a saved search. For more information, see Saved search.

6.Enable Dedicated SQL

Dedicated SQL is a paid feature that is provided by Log Service. You can use the Dedicated SQL feature to analyze log data by using SQL statements. Compared with the Standard SQL feature, which you can use for free, the Dedicated SQL feature has no limits on the number of concurrent operations or the amount of data to be analyzed.

Context

If you use the Standard SQL feature to analyze a large amount of log data that is generated over a period of time, Log Service cannot scan all log data in a single query. To ensure timeliness, Log Service limits the amount of data that is scanned in each shard and returns some inaccurate results. In this case, we recommend that you increase the number of shards to increase computing resources. However, after you increase the number of shards, only new data that is written to the shards can be read for scanning. Historical data cannot be read for scanning. The number of consumers also increases.

To resolve this issue, Log Service provides the Dedicated SQL feature. The Dedicated SQL feature can efficiently analyze log data and is not subject to the resource limits that compromise the performance of the Standard SQL feature. For more information, see Limits.

> **Note** The Dedicated SQL feature and the Standard SQL feature are both available. You can choose between the features based on your business requirements.

Advantages

You can use the Dedicated SQL feature to analyze log data by using SQL statements. The Dedicated SQL feature has the following advantages over the Standard SQL feature:

- The Dedicated SQL feature can analyze hundreds of billions of data records with high performance.
- The Dedicated SQL feature allows up to 100 concurrent operations in each project. The Standard SQL feature allows only 15 concurrent operations.
- The Dedicated SQL feature is allocated exclusive resources. The performance of the Dedicated SQL feature is not affected by traffic bursts from other users.

Scenarios

The Dedicated SQL feature is suitable for the following scenarios:

- You need to analyze data with high performance. For example, you need to analyze data in real time.
- You need to analyze data that is generated over a long period of time. For example, you need to analyze data that is generated over a month.
- You need to analyze a large amount of data. For example, you need to analyze terabytes of data every day.
- You need to analyze data by using more than 15 concurrent SQL statements and display the analysis results based on multiple metrics from multiple dimensions.

Procedure

1.
2.
3.
4. Click the icon.

   After you enable the Dedicated SQL feature, you can use this feature to query and analyze log data by using SQL statements. For more information, see Query and analyze logs.

> **Notice** If you enable the Dedicated SQL feature for a Logstore of a project, the analysis and query operations in other Logstores of the project are not affected.

SDK examples

- Use Log Service SDK for Java to use Dedicated SQL
- Use Log Service SDK for Python to use Dedicated SQL
- Use Log Service SDK for Node.js to use Dedicated SQL
- Use Log Service SDK for PHP to use Dedicated SQL
- Use Log Service SDK for C++ to use Dedicated SQL

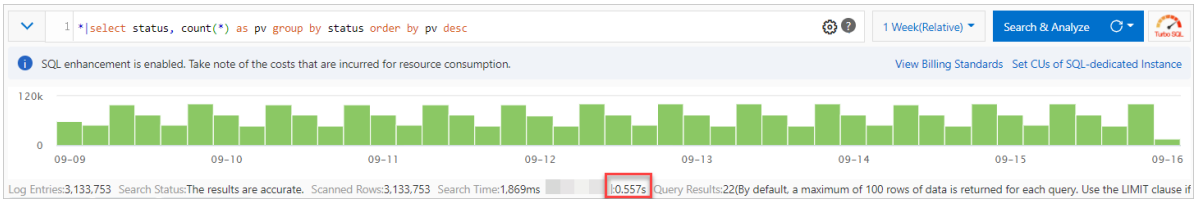
FAQ

- How do I enable the Dedicated SQL feature by calling an API operation?

  You can enable the Dedicated SQL feature by calling the GetLogs operation. When you call this operation, you can use the powerSql or query parameter to specify whether to enable the Dedicated SQL feature. For more information, see GetLogs.

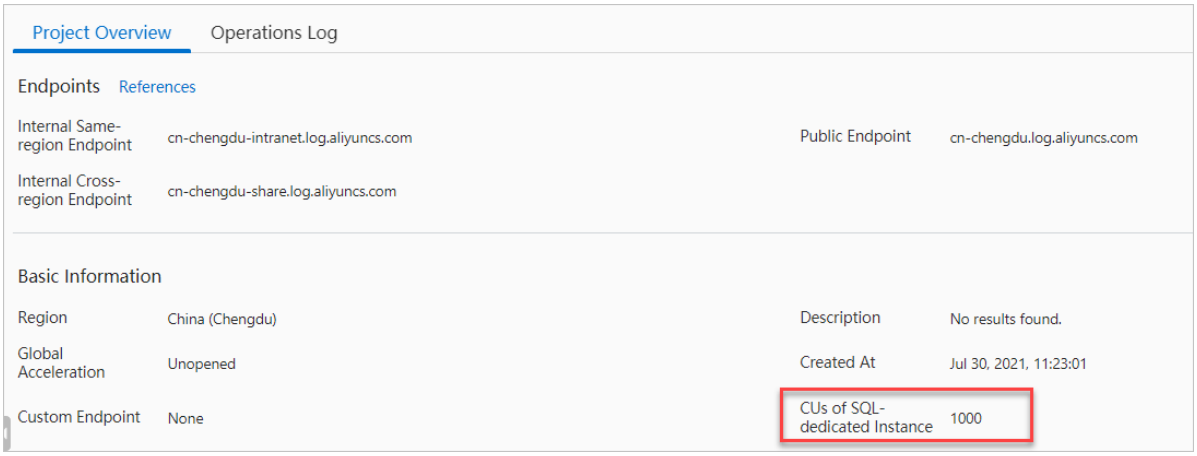
- How do I obtain the amount of CPU time that I use?

  After you perform analysis and query operations, you can obtain the amount of CPU time that you use in the Log Service console. The following figure shows an example.



- Can I control the cost of the Dedicated SQL feature?

  Yes, you can modify the number of CUs to control the cost of the Dedicated SQL feature. To modify the number of cores, go to the **Project Overview** page of your project and change the value of the **CUs of SQL-dedicated Instance** parameter.



- What are the fees of the Dedicated SQL feature when I execute a query statement once?

  The fees of the Dedicated SQL feature vary based on the amount of data on which you execute query statements. The following table provides examples.

Query statement	Amount of data (rows)	Average cost per execution (USD)
`* | select avg(double_0) from stress_s1_mil1`	4 billion	0.004435
`* | select avg(double_0), sum(double_0), max(double_0), min(double_0), count(double_0) from stress_s1_mil1`	4 billion	0.006504
`* | select avg(double_0), sum(double_1), max(double_2), min(double_3), count(double_4) from stress_s1_mil1`	4 billion	0.013600
`* | select key_0 , avg(double_0) as pv from stress_s1_mil1 group by key_0 order by pv desc limit 1000`	4 billion	0.011826
`* | select long_0, avg(double_0) as pv from stress_s1_mil1 group by long_0 order by pv desc limit 1000`	4 billion	0.011087
`* | select long_0, long_1, avg(double_0) as pv from stress_s1_mil1 group by long_0,long_1 order by pv desc limit 1000`	0.3 billion	0.010791
`* | select avg(double_0) from stress_s1_mil1 where key_0='key_987'`	4 billion	0.00007

7.Query

7.1. Search syntax

This topic describes the search syntax of Log Service. You can configure query conditions by using the search syntax.

Search types

A search statement specifies one or more query conditions and returns the logs that match the specified conditions. Searches are classified into full-text searches and field-specific searches based on the indexing method, or classified into exact searches and fuzzy searches based on precision.

> **Note**
> - If you configure both full-text indexes and field indexes, the configurations of the field indexes take precedence for operations.
> - Before you can specify a numeric range to query logs based on a field, you must set the data type of the field to double or long in the index configurations. If you do not set the data type of the field to double or long or the syntax of the numeric range is invalid, Log Service performs a full-text search, and the search result may be different from the expected result. For example, if you execute the `owner_id>100` search statement and the data type of the owner_id field is not double or long, logs that contain owner_id, >, and 100 are returned. In this example, the greater sign (>) is not a delimiter.
> - If you change the data type of a field from text to double or long in the index configurations, you can use only the equal sign (=) to query the logs that are collected before the change takes effect.

- Full-text searches and field-specific searches

Search type	Description	Example
Full-text search	After you configure full-text indexes, Log Service splits a log into multiple words by using the delimiters that you specify. You can specify keywords and rules in a search statement to query logs. The keywords can be field names or field values.	`PUT and cn-shanghai` : returns the logs that contain the keywords PUT and cn-shanghai.
Field-specific search	After you configure field indexes, you can query logs by specifying field names and field values in the key:value format. You can perform basic searches or combined searches based on the data types of fields in the index configurations. For more information, see Data types.	`request_time>60 and request_method:Ge*` : returns the logs in which the value of the request_time field is greater than 60 and the value of the request_method field starts with Ge.

- Exact searches and fuzzy searches

Search type	Description	Example
Exact search	Complete words are used for searches.	◦ `host:example.com` : returns the logs in which the value of the host field is example.com. ◦ `PUT` : returns the logs that contain the keyword PUT.

Search type	Description	Example
Fuzzy search	You can add an asterisk (*) or a question mark (?) as a wildcard character to the middle or end of a word in a search statement when you perform a fuzzy search. The word must be 1 to 64 characters in length. If a word contains a wildcard character, Log Service searches all logs and obtains up to 100 words that match the word. Then, Log Service returns the logs that contain one or more of these words. If you specify more accurate words, the search result is more accurate. **Note** ◦ You cannot add an asterisk (*) or a question mark (?) to the start of a word. ◦ The long and double data types do not support asterisks (*) or question marks (?) in fuzzy searches. You can specify a numeric range when you perform a fuzzy search. Example: status in [200 299]. A fuzzy search is performed based on samples by using the following mechanism: ◦ If you enable the field indexing feature and specify a field to query logs, Log Service obtains random samples from the indexed data of the field for scanning and returns results. Log Service does not perform full-text scans. ◦ If you enable the full-text indexing feature and you do not specify a field to query logs, Log Service obtains random samples from the full-text indexed data and returns results. Log Service does not perform full-text scans.	◦ `addr*` : obtains 100 words that start with addr from all logs, and returns the logs that contain one or more of these words. ◦ `host:www.yl*` : obtains 100 words that start with www.yl from the values of the host field in all logs, and returns the logs that contain one or more of these words. For more information, see Fuzzy match.

Operators

The following table describes the operators that are supported for search statements.

Note

- The in operator is case-sensitive. Other operators are not case-sensitive.
- Log Service supports the following operators: **sort**, **asc**, **desc**, **group by**, **avg**, **sum**, **min**, **max**, and **limit**. If you want to use these operators as keywords, you must enclose the operators in double quotation marks ("").
- The following list shows the priorities of operators in descending order:
  i. Colons (:)
  ii. Double quotation marks ("")
  iii. Parentheses ()
  iv. and, not
  v. or

Operator	Description
and	The and operator. Example: `request_method:GET and status:200`. If no syntax keywords exist among multiple keywords, the keywords are evaluated by using the and operator. For example, `GET 200 cn-shanghai` is equivalent to `GET and 200 and cn-shanghai`.
or	The or operator. Example: `request_method:GET or status:200`.
not	The not operator. Examples: `request_method:GET not status:200` and `not status:200`.
( )	This operator is used to increase the priority of the query conditions that are enclosed in parentheses (). Example: `(request_method:GET or request_method:POST) and status:200`.
:	This operator is used for field-specific searches based on the key:value format. Example: `request_method:GET`. If a field name or a field value contains reserved characters such as spaces and colons (:), you must enclose the field name or field value in double quotation marks (""). Example: `"file info":apsara`.
""	This operator is used to enclose a syntax keyword. If a syntax keyword is enclosed in double quotation marks (""), the keyword is converted to an ordinary character. For example, `"and"` returns the logs that contain and. In this case, and is not an operator. In a field-specific search, the words that are enclosed in double quotation marks ("") are considered as a whole.
\	The escape character. This character is used to escape double quotation marks (""). Double quotation marks ("") can indicate themselves only after they are escaped. For example, if the content of a log is `instance_id:nginx"01"`, you can execute the `instance_id:nginx\"01\"` statement to search for the log.
*	The wildcard character. This character is used to match zero, one, or multiple characters. Example: `host:aliyund*c`. **Note** Log Service searches all logs and obtains up to 100 words that match the specified conditions. Then, Log Service returns the logs that contain one or more of these words and match the query conditions.
?	The wildcard character. This character is used to match a single character. Example: `host:aliyund?c`.
>	This operator is used to query the logs in which the value of a specified field is greater than a specified numeric value. Example: `request_time>100`.
>=	This operator is used to query the logs in which the value of a specified field is greater than or equal to a specified numeric value. Example: `request_time>=100`.
<	This operator is used to query the logs in which the value of a specified field is smaller than a specified numeric value. Example: `request_time<100`.

Operator	Description
<=	This operator is used to query the logs in which the value of a specified field is smaller than or equal to a specified numeric value. Example: `request_time<=100`.
=	This operator is used to query the logs in which the value of a specified field is equal to a specified numeric value. Equal signs (=) and colons (:) have the same effect on fields of the double or long data type. For example, `request_time=100` is equivalent to `request_time:100`.
in	This operator is used to query the logs in which the value of a specified field is within a specified numeric range. Brackets [] indicate a closed interval, and parentheses () indicate an open interval. A space is used to separate two numbers in a numeric range. Examples: `request_time in [100 200]` and `request_time in (100 200]`. **Note** The characters of in must be in lowercase.
__source__	This operator is used to query the logs of a specified log source. Wildcard characters are supported. Example: `__source__:192.0.2.*`. **Notice** The __source__ field is a reserved field in Log Service. This field can be abbreviated to source. If you configure a custom source field, the custom field conflicts with the reserved field source in Log Service. If you want to search for the custom field, you must use Source or SOURCE in the search statement.
__tag__	This operator is used to query logs by using metadata. Example: `__tag__:__receive_time__:1609837139`.
__topic__	This operator is used to query the logs of a specified log topic. Example: `__topic__:nginx_access_log`.

Examples of search statements

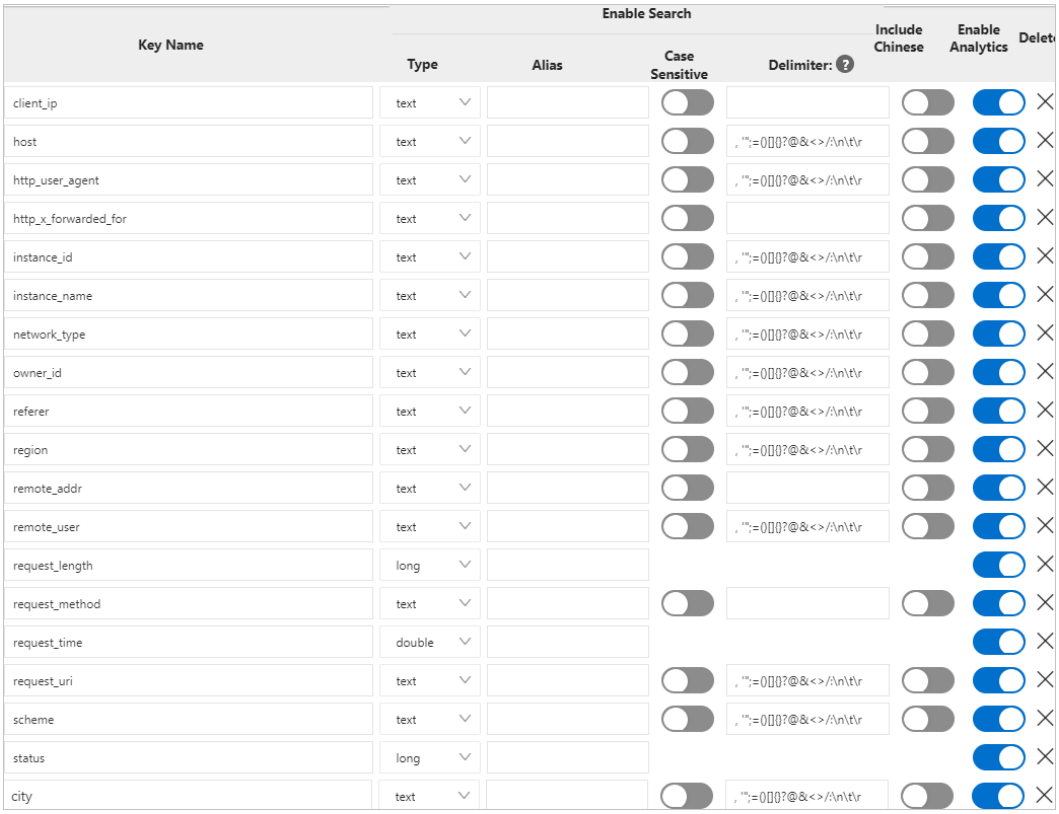
If you execute a search statement on different logs based on different index configurations, the statement returns different results. The examples in this section are provided based on the following sample log and index configurations.

Sample log

An NGINX access log is used as the sample log.

Index configurations

Before you can execute a search statement, make sure that indexes are configured. For more information, see Configure indexes.



Common search examples

Expected search result	Search statement
Logs that record successful GET requests (status codes: 200 to 299)	`request_method:GET and status in [200 299]`

Expected search result	Search statement
Logs that record GET requests and in which the source region of the requests is not the China (Hangzhou) region	`request_method:GET not region:cn-hangzhou`
Logs that record GET requests or POST requests	`request_method:GET or request_method:POST`
Logs that do not record GET requests	`not request_method:GET`
Logs that record successful GET requests or successful POST requests	`(request_method:GET or request_method:POST) and status in [200 299]`
Logs that record failed GET requests or failed POST requests	`(request_method:GET or request_method:POST) not status in [200 299]`
Logs that record successful GET requests (status codes: 200 to 299) and in which the request duration is less than 60 seconds	`request_method:GET and status in [200 299] not request_time>=60`
Logs in which the request duration is equal to 60 seconds	`request_time:60` `request_time=60`
Logs in which the request duration is greater than or equal to 60 seconds and is less than 200 seconds	`request_time>=60 and request_time<200` `request_time in [60 200)`
Logs in which the request_time field is empty or the value of the field is an invalid number	`request_time:* not request_time > -10000000000`
Logs that contain the request_time field and in which the value of the field is a number	`status > -1000000000`
Logs that contain and	`"and"` **Note** In this search statement, and is a common string but not an operator.

Expected search result	Search statement
Logs in which the value of the request method field contains PUT	`"request method":PUT` **Note** The name of the request method field contains spaces. You must enclose the field name in double quotation marks ("") in a search statement.
Logs whose topic is HTTPS or HTTP	`__topic__:HTTPS or __topic__:HTTP`
Logs that are collected from the 192.0.2.1 host	`__tag__:__client_ip__:192.0.2.1` **Note** The `__tag__:__client_ip__` field is a reserved field in Log Service. The field indicates the IP address of the host from which logs are collected. For more information, see Reserved fields. If a log is processed by using the data transformation feature or a Logtail plug-in, the key in a tag field is converted to a common key. If you want to search for the log, you must enclose the name of the tag field in double quotation marks ("") in the search statement. Example: `"__tag__:__client_ip__":192.0.2.1`.
Logs in which the remote_user field is not empty	`not remote_user:""`
Logs in which the remote_user field is empty	`remote_user:""`
Logs in which the value of the remote_user field is not null	`not request_user:"null"`
Logs that do not contain the remote_user field	`not remote_user:*`
Logs that contain the remote_user field	`remote_user:*`
Logs in which the value of the city field is not Shanghai	`not city:Shanghai`

Advanced search examples

- Fuzzy searches

Expected search result	Search statement
Logs that contain certain words. The words start with cn.	`cn*`
Logs in which the value of the region field starts with cn.	`region:cn*`
Logs in which the value of the region field contains cn*	`region:"cn*"` **Note** In this search statement, `cn*` is a complete word. Examples: ◦ If the content of a log is `region:cn*,en` and the delimiter is a comma (,), Log Service splits the log content into `region`, `cn*`, and `en`. You can use the search statement to search for the log. ◦ If the content of a log is `region:cn*hangzhou`, Log Service considers `cn*hangzhou` as a whole. Therefore, you cannot use the search statement to search for the log.
Logs that contain certain words. The words start with mozi, end with la, and include one character between mozi and la.	`mozi?la`
Logs that contain certain words. The words start with mo, end with la, and include zero, one, or more characters between mo and la.	`mo*la`
Logs that contain certain words. The words start with moz or sa.	`moz* and sa*`
Logs in which the value of the region field ends with hai.	You cannot use a search statement to search for the logs. You can use the LIKE clause in an SQL statement to search for the logs. For more information, see Use the LIKE clause to implement fuzzy match. `* | select * from log where region like '%hai'`

- Delimiter-based searches

  Log Service splits the content of a log into multiple words based on the delimiters that you specify. The delimiters are `, '";=()[]{}?@&<>/:\n\t\r`. If you do not specify delimiters, Log Service considers the value of each field as a whole. In this case, you can search for a log only by using a complete string or a fuzzy search. For more information about how to specify delimiters, see Configure indexes.

  For example, the value of the http_user_agent field is `Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.2 (KHTML, like Gecko) Chrome/192.0.2.0 Safari/537.2`.

  ◦ If you do not specify delimiters, Log Service considers the field value as a whole. In this case, you cannot search for logs by using the `http_user_agent:Chrome` search statement.

- If you specify delimiters, Log Service splits the field value into `Mozilla`, `5.0`, `Windows`, `NT`, `6.1`, `AppleWebKit`, `537.2`, `KHTML`, `like`, `Gecko`, `Chrome`, `192.0.2.0`, `Safari`, and `537.2`. The delimiters are `, '";=()[]{}?@&<>/:\n\t\r`. In this case, you can search for logs by using the `http_user_agent:Chrome` search statement.

Expected search result	Search statement
Logs in which the value of the http_user_agent field contains Chrome	`http_user_agent:Chrome`
Logs in which the value of the http_user_agent field contains Linux and Chrome	`http_user_agent:"Linux Chrome"` `http_user_agent:Linux and http_user_agent:Chrome`
Logs in which the value of the http_user_agent field contains Firefox or Chrome	`http_user_agent:Firefox or http_user_agent:Chrome`
Logs in which the value of the request_uri field contains /request/path-2	`request_uri:/request/path-2`
Logs in which the value of the request_uri field starts with /request and does not contain /file-0	`request_uri:/request* not request_uri:/file-0`

7.2. Phrase search

This topic describes the syntax and limits of the phrase search feature and provides usage examples.

Overview

Notice Only the following regions support the phrase search feature: Russia (Moscow), UK (London), Philippines (Manila), and Malaysia (Kuala Lumpur).

Log Service uses the word segmentation method to query specific logs. For example, the search statement `abc def` returns logs that contain `abc` or `def`. In addition, the results are sorted in random order and the specified phrases cannot be exactly matched. Log Service allows you to use the phrase search feature to perform exact phrase searches.

After you send a request to query logs that contain a phrase, Log Service performs the following steps:

1. Execute the non-phrase part of a search statement to query logs. For example, if you execute the search statement `#"/92//docvalue"`, Log Service first executes the `"/92//docvalue"` statement.

   Note Up to 10,000 logs can be returned by the phrase search statement in Step 1. This prevents Log Service from handling an excessive amount of data within a short period of time.

2. Query the logs that match the search condition from the preceding search results, and then return the final search results.

Syntax

- Field-specific search

```
key:#"abc def"
```

- Full-text search

```
#"abc def"
```

Limits

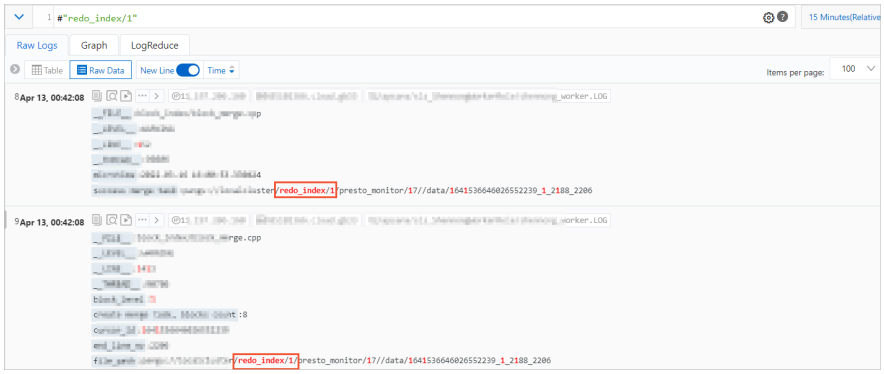
- The results of a phrase search can be paged only forward or backward based on the sequence of page numbers. Random redirection is not supported.
- After you perform a phrase search, the log distribution histogram shows the distribution of non-phrase search results.
- Fuzzy match is not supported in phrase searches.
- In a phrase search statement, you must enclose the phrases that you want to match within a pair of quotation marks (“”).
- The NOT operator is not supported in phrase search statements. Example: `not #"abc def"` .
- Analytic statements are not supported in phrase searches. Example: `#"abc" | select ***` . The quick analysis feature is not supported when you perform phrase searches.

Examples

Query the logs that contain `redo_index/1` .

- If you execute the non-phrase part of the search statement `"redo_index/1"` , Log Service searches for logs that contain specific keywords based on full-text indexes.

- If you execute the phrase search statement `#"redo_index/1"` , Log Service searches for logs that contain the complete phrase `redo_index/1` .



7.3. LiveTail

This topic describes how to monitor and analyze log data by using LiveTail.

Prerequisites

Log data is collected by Logtail. For more information, see Use Logtail to collect data.

> **Notice** LiveTail can monitor and analyze only the log data that is collected by Logtail.

Context

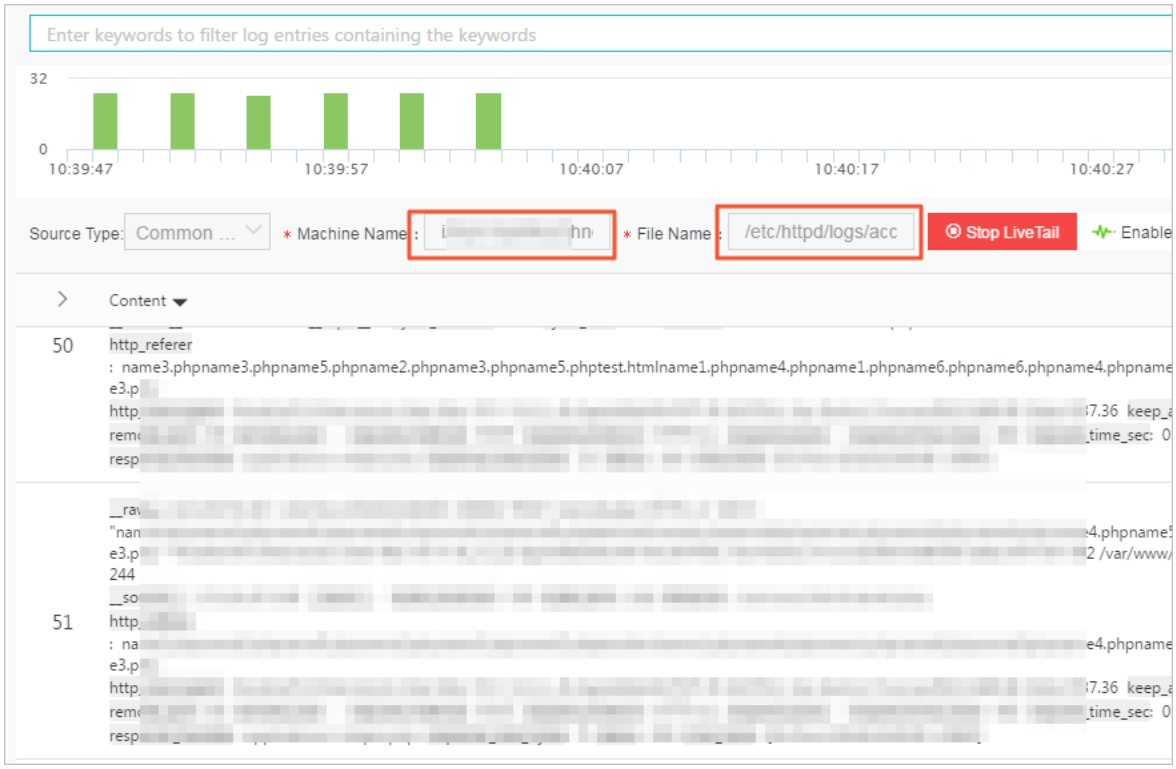
In online O&M scenarios, you may need to monitor log data in real time and extract crucial information from the latest log data to troubleshoot exceptions. If you use a traditional O&M method, you must run the **tail -f** command on each server to query log data. If you want to narrow the scope of the command output, you must run the **grep** or **grep -v** command to filter the log data by keyword. To simplify online O&M operations, Log Service provides LiveTail in the Log Service console. You can use LiveTail in the Log Service console to monitor and analyze log data in real time.

Benefits

- Log entries are monitored in real time and can be filtered by keyword.
- Log entries are collected and indexed based on the collection configuration.
- The content of log fields is segmented into words. The word segmentation feature allows you to query contextual log entries that contain specific words.
- You can query a specified log entry from the log file that contains the log entry. This way, you can monitor the log entry in the log file in real time without the need to log on to a server.

Procedure

1.
2.
3.
4. Click the **Raw Logs** tab. On the Raw Logs tab, click the **Raw Data** button. Then, click the icon for a log entry to start LiveTail.
5. In the **LiveTail** section, view log entries.
   After LiveTail is started, the log entries that are collected by Logtail are displayed in the LiveTail section in real time. By default, the latest log entries are displayed at the bottom of LiveTail section. You can view the latest log entries without the need to scroll down. A maximum of 1,000 log entries can be displayed. If more than 1,000 log entries are collected, the LiveTail section is automatically refreshed to display the latest 1,000 log entries.



More operations

Operation	Description
Highlight strings	You can enter one or more strings in the **Highlight** field. The specified strings are highlighted in the LiveTail section.
Filter log entries by string	You can enter one or more strings in the **Filter By** field. The LiveTail section displays only the log entries that contain the specified strings.
Filter log entries by field	You can select one or more fields from the **Filter by Field** drop-down list. The LiveTail section does not display the log entries that contain the specified fields.
Stop LiveTail	You can click **Stop** to stop LiveTail. After LiveTail is stopped, the LiveTail section is no longer refreshed. You can analyze the log entries in the LiveTail section to troubleshoot exceptions.

7.4. LogReduce

This topic describes how to use the LogReduce feature of Log Service. You can enable the feature, view log clustering results and raw logs, adjust clustering precision, and compare the number of log entries in different time periods.

Context

The LogReduce feature allows you to cluster similar logs and extract patterns from the logs. The feature supports text logs of multiple formats. You can use the feature to locate errors, detect anomalies, roll back versions, and perform other O&M operations in DevOps scenarios. You can also use the feature to detect network intrusions to ensure data security. In addition, you can save log clustering results as charts to a dashboard, and then view the clustered data in real time.

Benefits

- You can cluster logs of multiple formats, such as Log4j logs, JSON-formatted logs, and single-line logs.
- Hundreds of millions of log entries can be clustered in seconds.
- Log data can be clustered in multiple modes.
- Raw log entries can be retrieved based on pattern signatures.
- You can compare log patterns of different time periods.
- You can adjust the clustering precision of logs.

Indexes

If you enable the LogReduce feature, the size of indexes increases by 10% of the raw log size. For example, if the size of raw log data is 100 GB per day, the size of indexes increases by 10 GB.

Raw log size	Index percentage	Size of indexes generated by LogReduce	Index size
100 GB	20% (20 GB)	100 * 10%	30 GB
100 GB	40% (40 GB)	100 * 10%	50 GB
100 GB	100% (100 GB)	100 * 10%	110 GB

Enable the LogReduce feature of a Logstore

1.
2.
3.
4. Enable the LogReduce feature of a Logstore.
   i. Choose **Index Attributes > Attributes**.

      If the indexing feature is not enabled, click **Enable**.
   ii. In the **Search & Analysis** dialog box, turn on the **LogReduce** switch.
   iii. (Optional)Configure a whitelist or blacklist to filter fields.

      You can filter logs by keyword. Logs that are filtered based on keywords are automatically clustered.
   iv. Click **OK**.

View log clustering results and raw logs

1. On the Search & Analysis page, enter a search statement in the search box, and then click **Search & Analyze**.

   > **Note** You can use only search statements to filter logs. However, you cannot use analytic statements to analyze logs because the LogReduce feature cannot cluster analysis results.

2. Click the **LogReduce** tab to view the log clustering results.

   You can also click **Add to New Dashboard** to save the log clustering result to a dashboard.

Parameter	Description
Number	The sequence number of a log cluster.
Count	The number of log entries of a pattern. The log entries are obtained in the current time range.
Pattern	The log pattern. Each log cluster has one or more sub-patterns.

- Move the pointer over a value in the **Count** column to view the sub-patterns of the corresponding log cluster. You can also view the percentage of each sub-pattern in the log cluster. Click the plus sign (+) next to the value to expand the sub-pattern list.
- Click a value in the **Count** column. You are redirected to the **Raw Logs** tab. On this tab, you can view the raw logs of the corresponding pattern.

Adjust the precision of log clustering

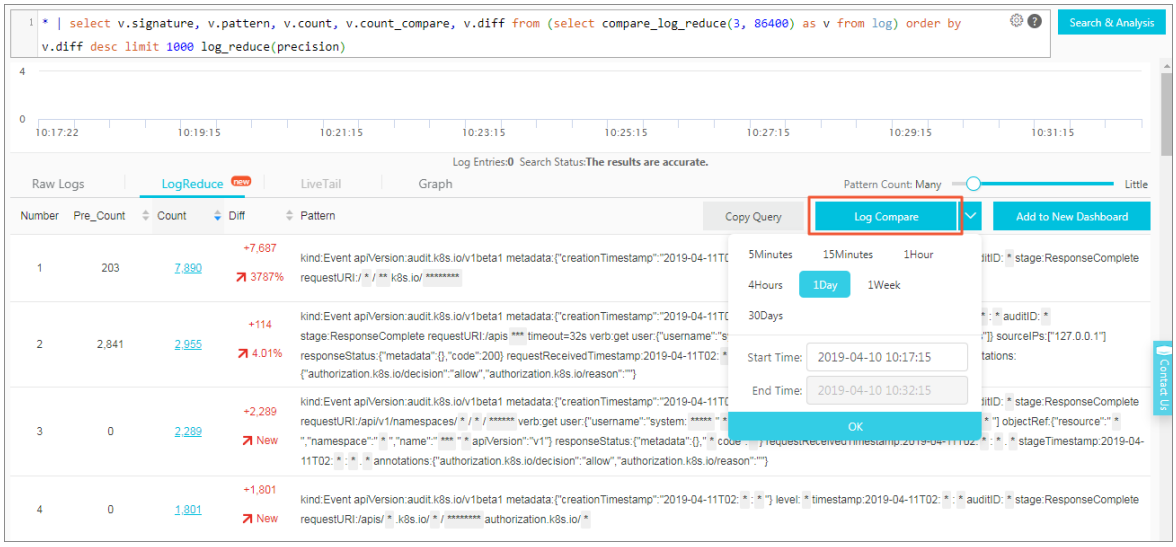
On the **LogReduce** tab, you can drag the **Pattern Count** slider to adjust the clustering precision.

- If you drag the slider towards **Many**, you can obtain a more precise log clustering result with more detailed patterns.
- If you drag the slider towards **Little**, you can obtain a less precise log clustering result with fewer detailed patterns.

Compare the number of log entries that are clustered in different time periods

1. On the **LogReduce** tab, click **Log Compare**.
2. Set a time range, and then click **OK**.

   For example, assume you set the time range to 15 minutes when you perform a query, and select **1 Day** for **Log Compare**. The start time and end time for log comparison are displayed. The time range is 15 minutes.



Parameter	Description
Number	The sequence number of a log group.
Pre_Count	The number of log entries of a pattern that are obtained in the time range specified by **Log Compare**.
Count	The number of log entries of a pattern. The log entries are obtained in the current time range.
Diff	The difference between the number of log entries that are specified by the Pre_Count and Count parameters and the growth rate.
Pattern	The log pattern.

Examples

You can use query statements to obtain clustered log data.

- Obtain log clustering results.
  - Query statement

```
* | select a.pattern, a.count,a.signature, a.origin_signatures from (select log_reduce(3) as a f
rom log) limit 1000
```

> **Note** When you view log clustering results, you can click **Copy Query** to obtain the query statement.

  - Parameter settings

    In the query statement, the log_reduce(precision) function indicates the clustering precision. Valid values: 1 to 16. A smaller value indicates a higher clustering precision and more patterns. Default value: 3.

- Returned fields

  You can view the clustering details on the **Graph** tab.

Parameter	Description
pattern	The log pattern.
count	The number of log entries of a pattern. The log entries are obtained in the current time range.
signature	The signature of the log pattern.
origin_signatures	The secondary signature of the log pattern. You can use this signature to query corresponding raw data.

- Compare the number of log entries that are clustered in different time periods.
  - Query statement

```
* | select v.pattern, v.signature, v.count, v.count_compare, v.diff from (select compare_log_reduce(3, 86400) as v from log) order by v.diff desc limit 1000
```

> **Note** When you use **Log Compare** to compare log clustering results in different time periods, you can click **Copy Query** to obtain the query statement.

  - Parameter settings

    Modify parameter settings in the compare_log_reduce(precision, compare_interval) function.

    - The precision parameter indicates the clustering precision. Valid values: 1 to 16. A smaller number indicates a higher clustering precision and more patterns. Default value: 3.
    - The compare_interval parameter indicates the time difference between the two time ranges when the number of log entries are compared. The value is a positive integer. Unit: seconds.

  - Returned fields

Parameter	Description
pattern	The log pattern.
count_compare	The number of log entries of a pattern that are obtained in the time range specified by Log Compare.
count	The number of log entries of a pattern. The log entries are obtained in the current time range.
diff	The difference between the number of log entries that are indicated by the count and count_compare fields.
signature	The signature of the pattern.

Disable LogReduce of a Logstore

You can disable LogReduce if you no longer need to use it.

1. On the Search and Analysis page of the Logstore for which you want to disable this feature, choose **Index Attributes > Attributes**.
2. Turn off the **LogReduce** switch.
3. Click **OK**.

7.5. Context query

This topic describes how to query the contextual log entries of a specified log entry in the Log Service console.

Prerequisites

- Logs are collected by Logtail. For more information, see Logtail overview.

  > **Note** The context query feature is supported only for log data that is collected by Logtail.

- The indexing feature is enabled and configured. For more information, see Configure indexes.

Context

To perform a context query, you must specify a source server, source file, and a log entry whose context you want to query. You can obtain the log entries before or after the specified log entry that is collected from the log file of the server. This helps you identify and resolve errors.

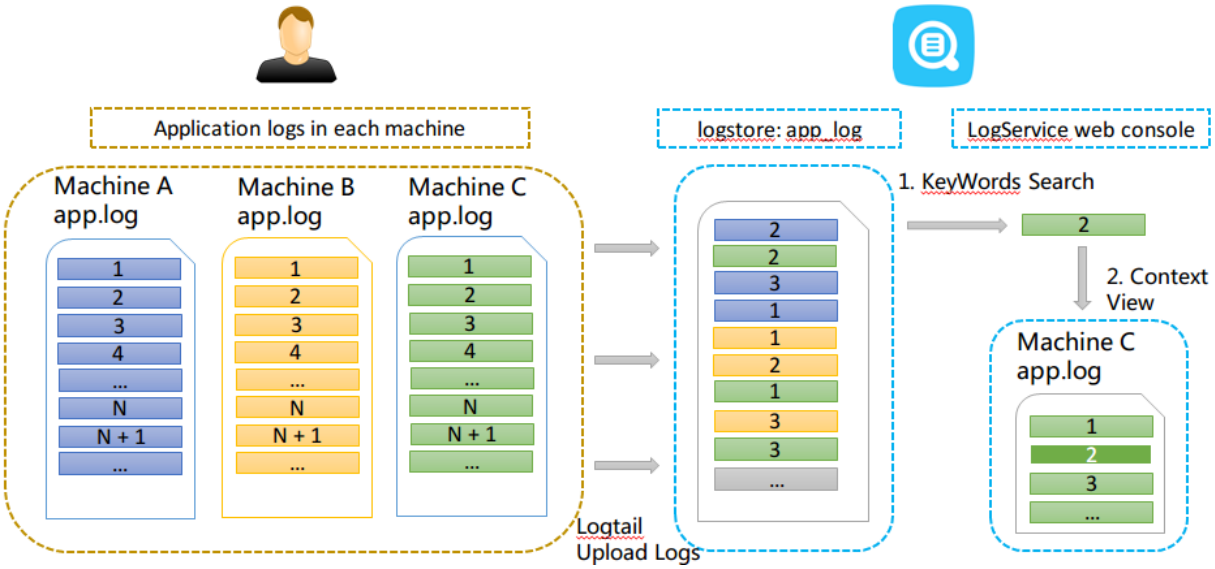
Scenarios

For example, an online-to-offline (O2O) website records the steps of an order into log entries on the server. These steps include: log on to the website, browse products, select a product, add the product to the shopping cart, place an order, pay for the order, deduct the order amount, and generate the order.

If an order fails, the O&M engineers must locate the cause at the earliest opportunity. If a traditional context query method is used, the O&M engineers must be authorized by an administrator to log on to each server where the O2O application is deployed. After authorization is complete, the O&M engineers must search the servers one by one for the related application log entries based on the order ID. Then, the O&M engineers can locate the failure cause based on the application log entries.

In Log Service, the O&M engineers can perform the following steps to locate the failure cause.

1. Install Logtail on the server. Create a machine group and configure a log collection file in the Log Service console. Then, enable Logtail to upload incremental log entries to Log Service.
2. On the search and analysis page in the Log Service console, specify a time range and find the log entry that records the failure based on the order ID.
3. After you locate the log entry, scroll up until other related log entries are found, for example, a log entry that records a credit card payment failure.

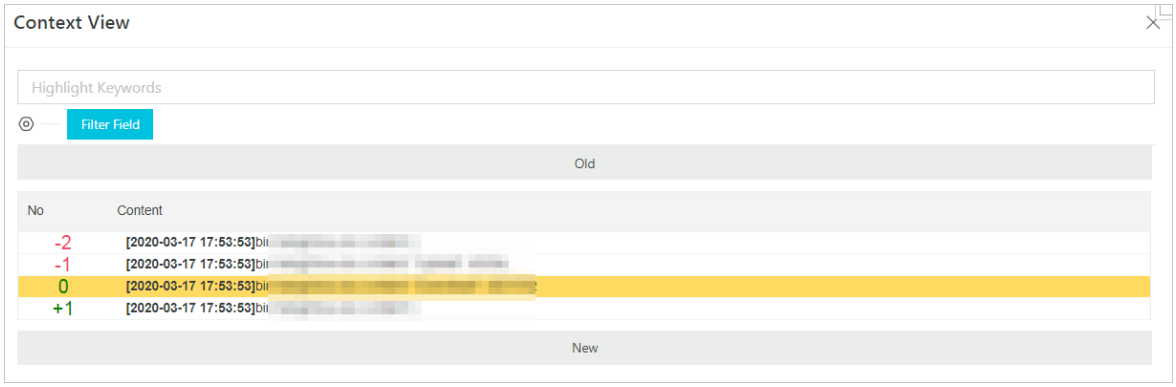


Benefits

- You do not need to change the format of application log entries.
- You can query the context of a log entry from a log file that is collected from a server in the Log Service console. You do not need to log on to the server to query the context.
- You can specify a time range to locate suspicious log entries before you perform a context query in the Log Service console. This improves troubleshooting efficiency.
- You do not need to worry about data loss that is caused by insufficient server storage or log file rotation. You can view log data history in the Log Service console at any time.

Procedure

1.
2.
3.
4. Enter a query statement, select a time range, and then click **Search & Analyze**.
5. On the **Raw Logs > Raw Data** tab, find the log entry whose context you want to query and click .
6. On the page that appears, scroll up and down to view the contextual log entries.
   - To scroll up, click **Old**.
   - To scroll down, click **New**.
   - To highlight a keyword, enter the keyword in the **Highlight** field.
   - To filter out fields, click **Filter By**. In the dialog box that appears, select the fields in the right pane and click Delete. After the fields are filtered out, the fields are removed from the log entries that are displayed in the Context View pane.



7.6. Saved search

If you need to frequently view the result of a query statement, you can save the query statement as a saved search. Log Service provides the saved search feature to save the required data query and analysis operations. You can use a saved search to quickly perform query and analysis operations.

Prerequisites

The indexing feature is enabled and indexes are configured. For more information, see Configure indexes.

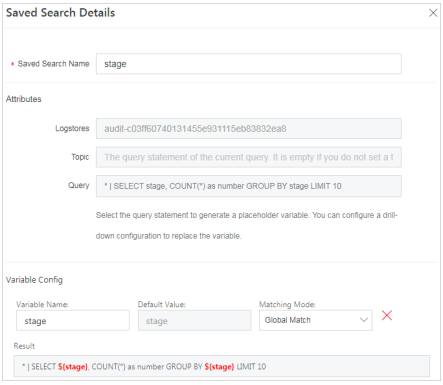
Create a saved search

1.
2.
3.

4. Enter a query statement in the search box, specify a time range, and then click **Search & Analyze**.

   A query statement consists of a search statement and an analytic statement in the format of Search statement|Analytic statement. For more information, see Search syntax and SQL syntax and functions.

5. Click **Save Search** in the upper-right corner of the page.
6. In the **Saved Search Details** panel, set the required parameters. The following table describes the parameters.



Parameter	Description
Saved Search Name	The name of the saved search.
Variable Config	To perform drill-down analysis, select the required content of the query statement in the **Query** field, and click **Generate Variables** to generate a placeholder variable. ○ **Variable Name**: the name of the placeholder variable. ○ **Default Value**: the content that you select from the **Query** field. ○ **Matching Mode**: the match mode. You can use the match mode to replace the default value by triggering a drill-down event. Valid values: Global Match and Exact Match. ■ Global Match: matches all words that are the same as the selected keyword in the query statement as variables. ■ Exact Match: matches the selected keyword in the query statement as a variable. For example, when you configure drill-down analysis for a chart, you set **Event Action** to **Open Saved Search** for the chart, and specify the saved search. The **Variable** of the chart is the same as the **Variable Name** of the saved search. When you click the chart value, you are redirected to the saved search. The **Default Value** of the placeholder variable is replaced by the chart value that triggers the drill-down event. Then, the new query statement is executed. For more information, see Configure a drill-down event for a chart. **Note** Before you set **Event Action** to **Open Saved Search**, you must create a saved search and configure variables.

7. Click **OK**.

   After you create a saved search, click the icon next to the search box on the Search & Analysis page of the Logstore, and click the name of the saved search to quickly perform query and analysis operations.

Modify a saved search

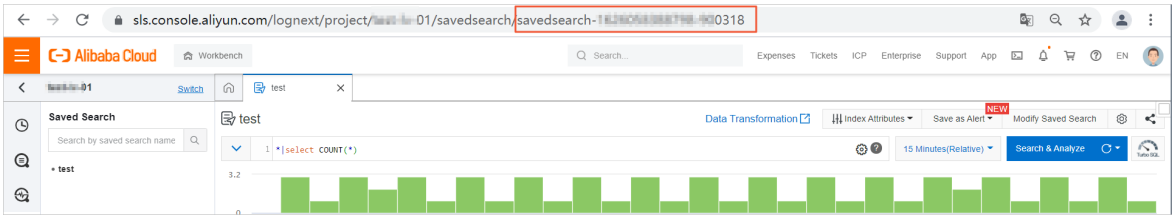
1. In the left-side navigation pane, choose **Resources > Saved Search**.
2. In the Saved Search list, click the saved search that you want to modify.
3. Enter a query statement and click **Search & Analyze**.

   A query statement consists of a search statement and an analytic statement in the format of Search statement|Analytic statement. For more information, see Search syntax and SQL syntax and functions.
4. Click **Modify Saved Search** in the upper-right corner of the page.
5. In the **Saved Search Details** panel, modify the required settings and click **OK**.

Obtain the ID of a saved search

After you create a saved search, you can use the ID of the saved search to embed the saved search page to a self-managed web page. For more information, see Customize the UI of embedded pages.

1. In the left-side navigation pane, choose **Resources > Saved Search**.
2. In the Saved Search list, click the saved search from which you want to obtain the ID.
3. Obtain the ID of the saved search in the URL.



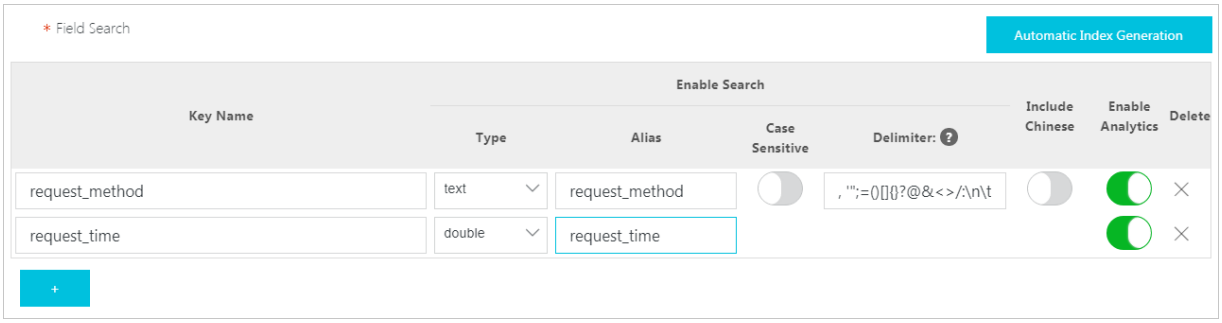
7.7. Quick analysis

The quick analysis feature of Log Service allows you to perform a query with ease. You can use this feature to analyze the distribution of a field over a period of time.

Prerequisites

Indexes are configured for specified fields. The analytics switch is turned on for these fields. For more information, see Configure indexes.

For example, if a log entry contains the request_method and request_time fields, you can configure indexes for the two fields. The following figure shows the configurations.



Features

- Allows you to analyze the first 100,000 log entries that are returned for a query.

  > **Note** When you perform a quick analysis during the selected time range, the first 100,000 log entries are returned. If you use a saved search to query all data in a Logstore, you must delete the Limit 100000 clause.

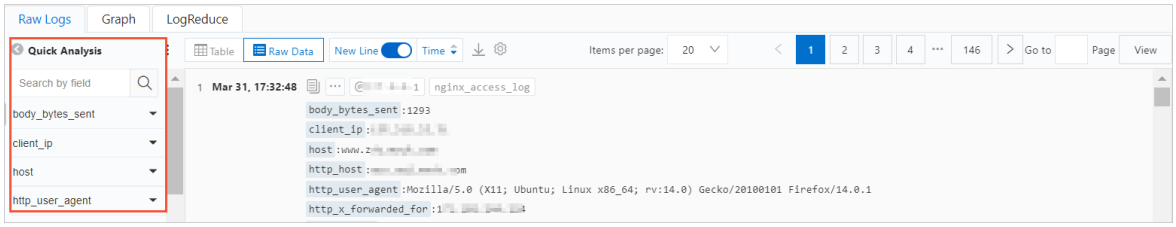
- Groups fields of the TEXT type and provides statistics about the top 10 groups.

- Generates **approx_distinct** statements for fields of the TEXT type.
- Supports histogram-based statistics about the approximate distribution of fields of the LONG and DOUBLE type.

  Histogram-based statistics groups sampling data and calculates the average value of each group.
- Searches for the maximum, minimum, average, or sum of fields of the LONG and DOUBLE type.
- Generates a query statement based on a quick analysis.

Procedure

1.
2.
3.
4. On the **Raw Logs** tab, click the destination field in the **Quick Analysis** column.



   - Provide grouping statistics for fields of the TEXT type and approximate distribution histogram-based statistics for fields of the LONG and DOUBLE type. For more information, see TEXT type or LONG and DOUBLE types.
   - Provide query statements.

     Click the icon next to the destination field. You are redirected to the **Graph** tab. A query statement for grouping statistics is provided in the search box.
   - Calculate the number of unique values of a field.

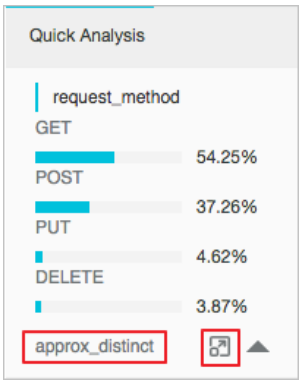
     In the Quick Analysis column, click **Count Distinct Values** under the destination field. You can obtain the number of unique values of the `${keyName}` field.
   - Display field names or aliases

     Click the icon to specify whether to display field names or aliases. Aliases can be set when you configure indexes. For example, if you set the alias of host_name to host, host is displayed in the Quick Analysis column after you select Show Field Aliases.

     > **Note** If you do not set an alias for a field, the field name is displayed after you select Show Field Aliases.

TEXT type

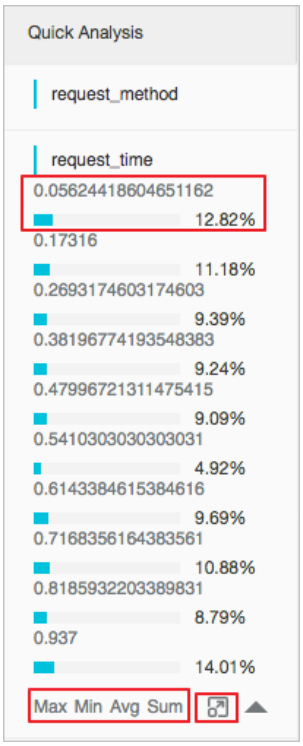
The quick analysis feature provides grouping statistics for fields of the TEXT type. If you use this method, the first 100,000 log entries are grouped and the ratios of the top 10 groups are returned. For example, you can obtain the following result based on grouping statistics of request_method. The GET method is the most common request method.



LONG and DOUBLE types

- Display approximate distribution by using histograms.

  The number of field values of the LONG and DOUBLE types is large. The preceding grouping analytics method is not suitable for the LONG or DOUBLE type. Log Service assigns field values into 10 buckets and displays the approximate distribution of the values in a histogram. The following figure shows the approximate distribution of the request_time field. This distribution of field value indicates that most of the request periods are distributed around 1.346 ms.



- Quick analysis on the maximum value, minimum value, average and sum of fields.

  You can click **Max** under a field to search for the maximum value, **Min** to search for the minimum value, **Avg** to calculate the average value, and **Sum** to calculate the sum of fields.

7.8. Reindex logs for a Logstore

Log Service allows you to reindex logs for a Logstore after you modify the current indexing rules or when you need to configure indexes for historical logs. You can reindex logs in a specified time range for a Logstore based on the latest indexing rules. This topic describes how to reindex logs for a Logstore in the Log Service console.

Limits

- You can reindex only the logs that are generated in the last 15 minutes to 30 days.
- You can create a maximum of 10 reindexing tasks.
- You can run only one reindexing task at a time.

Billing

- Reindexing incurs indexing fees and storage fees. Indexing fees are charged only once and storage fees are charged on an hourly basis. For more information, see Billable items.
- After you delete a reindexing task, the new index data is also deleted and no more storage fees are incurred for the deleted index data.

Procedure

1.
2.
3.
4. Click **Index Attributes > Reindex**.
5. In the **Reindex Logstore** dialog box, click **Create Task**.
6. In the dialog box that appears, set the **Task Name**, **Start Time** and **End Time** parameters, and then click **OK**.

   The **Start Time** is the time when Log Service starts to receive logs. The **End Time** is the time when Log Service stops receiving logs. The start time and end time must be in the last 15 minutes to 30 days.

   After you create the task, you can view the reindexing progress. When the progress reaches 100%, the reindexing task is completed.

What to do next

After you create a reindexing task, you can perform the following operations:

- Stop the reindexing task

  In the **Reindex Logstore** dialog box, click **Stop** to stop the reindexing task.

  > **Notice** After you stop the task, it can be deleted, but cannot be restarted. Proceed with caution.

- Delete the reindexing task

  In the **Reindex Logstore** dialog box, click **Delete** to delete the reindexing task.

  > **Notice** After you delete the reindexing task, the new index data is also deleted. Proceed with caution.

7.9. Configure events

Log Service allows you to configure drill-down events for raw logs to visualize logs and obtain more log details. You can configure default events and advanced events. This topic describes how to configure events for raw logs in the Log Service console.

Prerequisites

- The indexing feature is enabled and configured. For more information, see Configure indexes.
- A Logstore is created if you configure an advanced event to open a Logstore. For more information, see Create a Logstore.
- A saved search is created if you configure an advanced event to open a saved search. For more information, see Saved search.

  Placeholder variables are configured in the destination saved search if you configure variables. For more information, see Set a placeholder variable.
- A dashboard is created if you configure an advanced event to open a dashboard. For more information, see Create a dashboard.

  Placeholder variables are configured in the destination dashboard if you configure variables. For more information, see Set a placeholder variable.
- An HTTP link is created if you configure an advanced event to open a custom HTTP link.

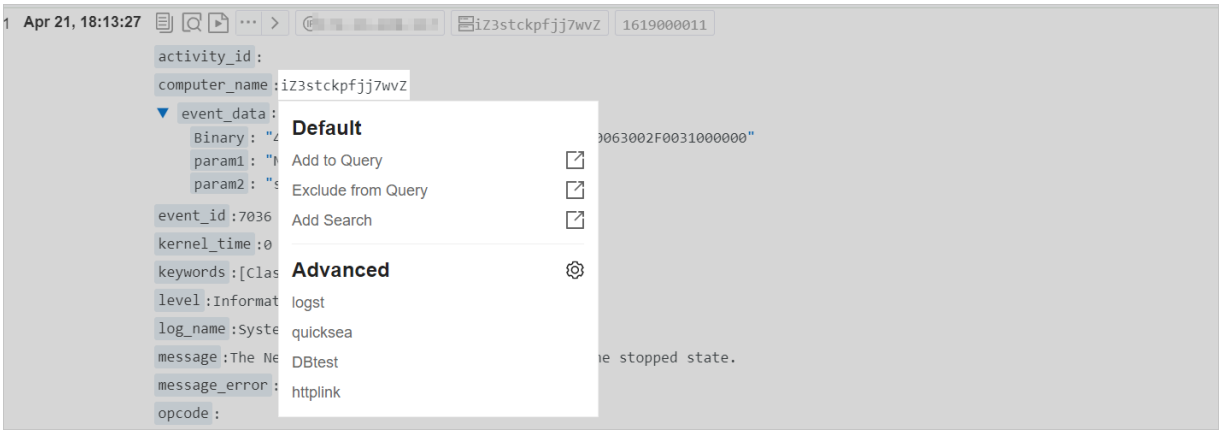
Context

Drilling is an essential feature in data analysis. This feature allows you to view more details by moving to different layers of data. Drilling includes rolling up and drilling down. Drilling down allows you to move to deeper data layers to gain an insight into data. This way, you can extract more value from data and make informative decisions. Log Service allows you to configure default events and advanced events to analyze raw logs.

Configure default events

When you configure default events, you can add conditions to query statements by using the AND and OR operators or create new query statements.

On the **Table** or **Raw Data** tab, click a field value. The **Default** dialog box appears. The following figure shows the operations that you can perform.



For example, the query statement you entered in the search box is `* | SELECT status as dim, count(1) as c group by dim`. If you click the value 203.0.113.1 in the host field, the query statement in the search box varies based on the event action you select.

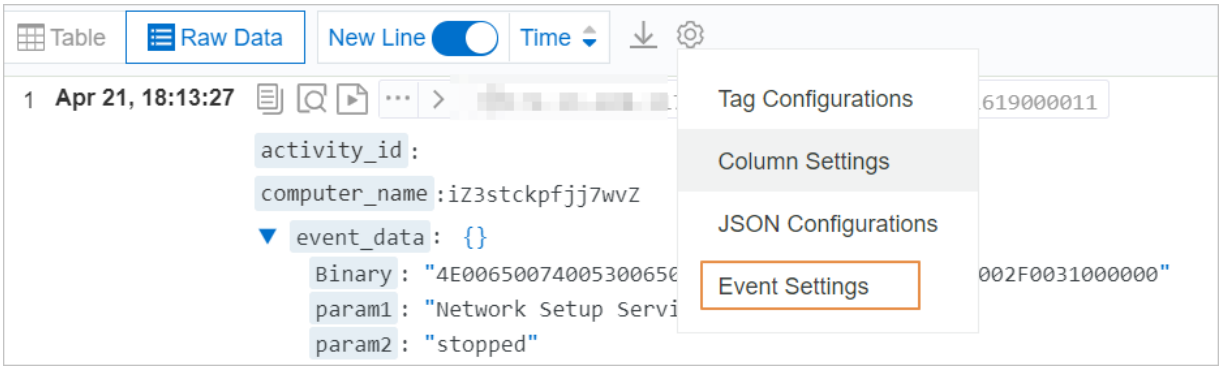
Event action	Description	Result
Add to Query	Append the keyword that you click to the query statement by using the AND operator and query the data.	`* and host: "203.0.113.1" | SELECT status as dim, count(1) as c group by dim`
Exclude from Query	Append the keyword that you click to the query statement by using the NOT operator.	`* not host: "203.0.113.1" | SELECT status as dim, count(1) as c group by dim`

Event action	Description	Result
Add Search	Delete the query statement from the search box and create a query statement by using the specified keyword.	`* and host: "203.0.113.1"`

Configure advanced events

You can configure advanced events for log fields to analyze logs at a deeper level. You can configure an advanced event to open a Logstore, saved search, dashboard, or a custom HTTP link.

On the **Table** or **Raw Data** tab, click  and then click **Event Settings** to go to the Advanced Event Settings window.



Note You can configure up to 10 advanced events for each log field.

1.
2.
3.
4. On the **Raw Logs** tab, click **Table** or **Raw Data**. Then, click  and **Event Settings** to go to the Advanced Event Settings window.
5. In the **Advanced Event Settings** window, click the field for which you want to add the advanced events, and click **Add Event**.
6. In the **Event Settings** section, set the required parameters.

   You can configure an advanced event to open a Logstore, saved search, dashboard, or a custom HTTP link. The following table describe the parameters.

   - **Open Logstore**

     Set the event action to open a Logstore. The following table describe the parameters.

Parameter	Description
Configuration Name	The name of the advanced event.
Event Action	Select **Open Logstore**.
Open in New Tab	If you turn on this switch, the specified Logstore is opened in a new tab when the advanced event is triggered.

Parameter	Description
Time Range	The time range that is used to query the data in the specified Logstore. Valid values: ■ **Default**: The time range that is used to query the data in the Logstore to which you are redirected. The default time range is 15 Minutes (Relative). ■ **Use Query Time**: The time range that is used to query the data in the Logstore to which you are redirected. This time range is the same as the time range specified to query raw logs. ■ **Relative**: The time range that is used to query the data in the Logstore to which you are redirected. This time range is accurate to the second. ■ **Time Frame**: The time range that is used to query the data in the Logstore to which you are redirected. This time range is accurate to the minute, hour, or day.
Select Logstore	The name of the Logstore to which you want to be redirected. When an advanced event is triggered, you are redirected to the Logstore page.
Inherit Filtering Conditions	If you turn on the **Inherit Filtering Conditions** switch, the filtering conditions of the current query are synchronized to the destination Logstore by using the `AND` operator.
Filter	If you enter a filter statement on the **Filter** tab, the filter statement is synchronized to the destination Logstore by using the `AND` operator. The filter statement can contain fields that you specify in the **Optional Parameter Fields** field. For example, if you click `${__topic__}`, the variable is appended to the query statement of the destination Logstore by using the `AND` operator.
Variable	Not supported.

- Open Saved Search

  Set the event action to open a saved search. The following table describes the parameters.

Parameter	Description
Configuration Name	The name of the advanced event.
Event Action	Select **Open Saved Search**.
Open in New Tab	If you turn on this switch, the specified saved search is opened in a new tab when the advanced event is triggered.
Time Range	The time range of the data that the saved search queries. Valid values: ■ **Default**: The time range of the data that the saved search queries is the default time range of a Logstore. The default time range is 15 Minutes (Relative). ■ **Use Query Time**: The time range of the data that the saved search queries is the same as the time range specified when you query raw logs. ■ **Relative**: The time range of the data that the saved search queries is accurate to the second. ■ **Time Frame**: The time range of the data that the saved search queries is accurate to the minute, hour, or day.

Parameter	Description
Select Saved Search	The name of the saved search to which you want to be redirected.
Inherit Filtering Conditions	If you turn on the **Inherit Filtering Conditions** switch, the filtering conditions of the current query are synchronized to the saved search to which you are redirected. The filtering conditions are appended to the saved search by using the `AND` operator.
Filter	If you enter a filter statement on the **Filter** tab, the filter statement is appended to the destination saved search by using the `AND` operator. The filter statement can contain fields that you specify in the **Optional Parameter Fields** field. For example, if you click `${__topic__}`, the variable is appended to the query statement of the destination saved search by using the `AND` operator.
Variable	Log Service allows you to modify a saved search by using variables. If you configure a variable that is the same as a variable in the saved search, the configured variable replaces the variable in the saved search. Click **Variable** to go to the Variable tab and configure variables. **Note** ■ If you configure variables for the event, you must first configure placeholder variables for the saved search to which you want to be redirected. For more information, see Set a placeholder variable. ■ You can add a maximum of five dynamic variables and five static variables. ■ Dynamic variables: The field value that you click to trigger the event is used as the variable for the query. ■ **Variable**: The name of the dynamic variable. For example, the placeholder variable that you specify in the saved search is `dynamic_ip`. ■ **Variable Value Column**: The column where the variable value is located. For example, if you select `__source__`, the value of the `__source__` field replaces the placeholder variable in the destination saved search. ■ Static variables: The static variable that you specify is used for the query. ■ **Variable**: The name of the static variable. For example, the placeholder variable that you specify in the saved search is `static_ip`. ■ **Static Value**: The value of the static variable that is used to replace the placeholder variable in the destination saved search. For example, `203.0.113.1` indicates that the value `203.0.113.1` of the `static_ip` field replaces the placeholder variable in the destination saved search. Logs whose placeholder variable value is `203.0.113.1` are queried.

- Open Dashboard

  Set the event action to open a dashboard. The following table describes the parameters.

Parameter	Description
Configuration Name	The name of the advanced event.
Event Action	Select **Open Dashboard**.
Open in New Tab	If you turn on this switch, the specified dashboard is opened in a new tab when the advanced event is triggered.
Time Range	The time range that is used to query for the dashboard to which you are redirected. Valid values: ■ **Default**: The time range that is used to query for the dashboard to which you are redirected is the default time range. The default time range is 15 Minutes (Relative). ■ **Use Query Time**: The time range of the chart on the destination dashboard is the time range of the chart that is specified on the source dashboard when the event is triggered. ■ **Relative**: The time range that is used to query for the dashboard to which you are redirected. This time range is accurate to the second. ■ **Time Frame**: The time range that is used to query for the dashboard to which you are redirected. This time range is accurate to the minute, hour, or day.
Select Dashboard	The name of the dashboard to which you want to be redirected.
Inherit Filtering Conditions	If you turn on the **Inherit Filtering Conditions** switch, the filtering conditions of the source dashboard are synchronized to the destination dashboard when the event is triggered.
Filter	If you enter a filter statement on the **Filter** tab, the filter statement is synchronized to the destination dashboard. The filter statement can contain fields that you specify in the **Optional Parameter Fields** field. For example, if you click `${__source__}`, only the logs that contain the `${__source__}` fields are displayed in the destination dashboard.

Parameter	Description
Variable	The variables that you configure are synchronized to the destination dashboard when the event is triggered. Click **Variable** to go to the Variable tab and configure variables. **Note** ■ If you configure variables for the event, you must first configure placeholder variables for the chart of the destination dashboard to which you want to be redirected. For more information, see Set a placeholder variable. ■ You can add a maximum of five dynamic variables and five static variables. ■ Dynamic variables: The field value that you click to trigger the event is used as the variable to query. ■ **Variable**: The name of the dynamic variable. For example, the placeholder variable that you specify in the destination dashboard is `dynamic_ip`. ■ **Variable Value Column**: The column where the variable value is located. For example, if you select `__source__`, the value of the `__source__` field replaces the placeholder variable in the destination dashboard. ■ Static variable: The static variable that you specify is used for the query. ■ **Variable**: The name of the static variable. For example, the placeholder variable that you specify in the destination dashboard is `static_ip`. ■ **Static Value**: The value of the static variable that is used to replace the placeholder variable in the destination dashboard. For example, `203.0.113.1` indicates that the value `203.0.113.1` of the `static_ip` field replaces the placeholder variable in the destination dashboard. Logs whose placeholder variable value is `203.0.113.1` are queried.

- Open HTTP Link

  Set the event action to open a custom HTTP link.

  - The path in the HTTP link is the path of the destination file.
  - If you add an optional parameter to the path and click a field value to trigger the advanced event, the added parameter is replaced by the field value. At the same time, you are redirected to the new HTTP link.

Parameter	Description
Configuration Name	The name of the advanced event.
Event Action	Select **Custom HTTP Link**.
Protocol	The protocol type that is used to access the custom HTTP link. You can select HTTP or custom protocol.

Parameter	Description
Enter Link	The address to which you want to be redirected. For example, if you enter `www.example.com/s?wd=${sls_project}`, you are redirected to this address after the event is triggered. The ${sls_project} parameter is replaced by the name of your project.
Use System Variables	If you turn on the **Use System Variables** switch, you can insert variables that are provided by Log Service into the HTTP link. The variables are ${sls_project}, ${sls_dashboard_title}, ${sls_chart_name}, ${sls_chart_title}, ${sls_region}, ${sls_start_time}, ${sls_end_time}, ${sls_realUid}, and ${sls_aliUid}.
Transcoding	If you turn on the **Transcoding** switch, the HTTP link is encoded.
Optional Parameter Fields	If you add an optional parameter to the path, the parameter is replaced by the field value when you click a field value to trigger the advanced event.

Example

The following example describes how to store access logs in a Logstore named accesslog. In this example, a saved search is created to query the page view (PV) distribution of IP addresses and request methods. On the Raw Logs page, set the advanced event for the remote_addr field to open a saved search. Then, click remote_addr. You are redirected to the saved search to view the PV distribution.

Raw log entry:

```
__source__:127.0.0.1
__tag__:__receive_time__:1613759995
__topic__:nginx_access_log
body_bytes_sent:5077
host:www.example.com
http_referer:www.example.com
http_user_agent:Mozilla/5.0 (X11; CrOS i686 12.0.742.91) AppleWebKit/534.30 (KHTML, like Gecko) Chrom
e/192.0.2.2 Safari/534.30
http_x_forwarded_for:192.0.2.1
remote_addr:192.0.2.0
remote_user:gp_02
request_length:3932
request_method:POST
request_time:35
request_uri:/request/path-2/file-4
status:200
time_local:19/Feb/2021:18:39:50
upstream_response_time:0.09
```

Procedure

1. Query the PV distribution of requests whose request method is POST and status code is 200. Create a saved search named **PV Distribution of IP Addresses and Request Method**. The following example shows the query statement and query result:

   ```
   * and request_method: POST and status: 200 | select count(*) as pv, remote_addr as ip,request_met
   hod as method group by ip,method order by ip desc
   ```

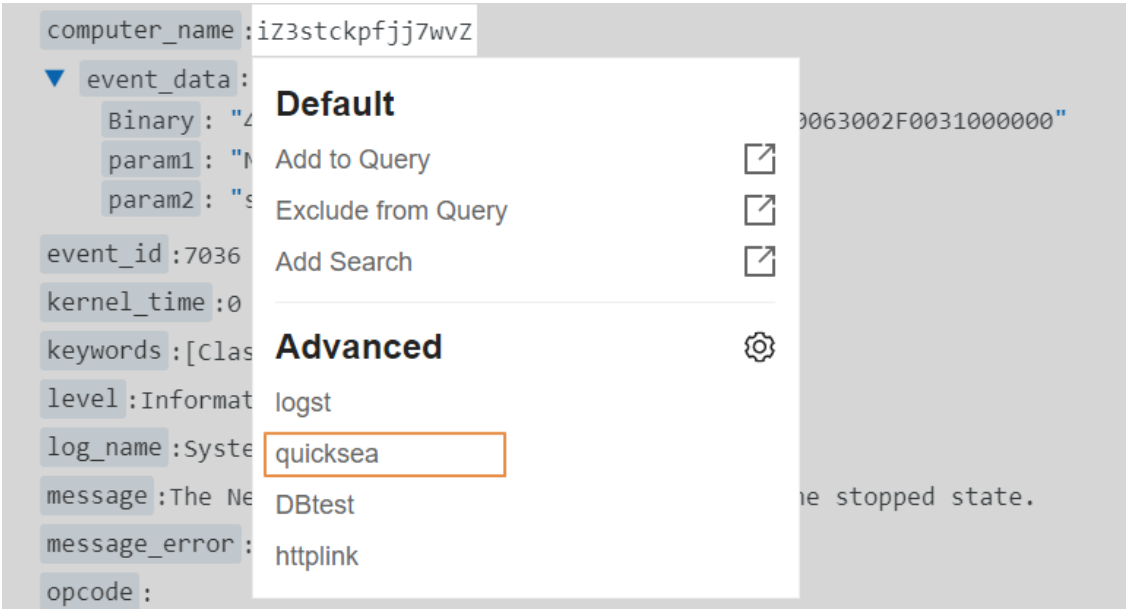
pv	ip	method
1	99 8	POST
1	99 6	POST
1	98 97	POST
1	98 9	POST
1	98 85	POST
1	98 77	POST

2. Set the `method` and `status2` variables in the query statement. The following example shows the query statement:

```
* and request_method: ${method} and status: ${status2} | select count(*) as pv, remote_addr as ip
,request_method as method group by ip,method order by ip desc
```

3. On the Raw Logs tab, set the advanced event for the remote_addr field to **Open Saved Search** and set the following parameters.
   - Select Quick Query: Select PV Distribution of IP Addresses and Request Method.
   - Filter: You do not need to specify the parameters on this tab.
   - Variables: Set the key of a static variable to status2 and the value to 400. Set the key of a dynamic variable to method and the value to request_method.
4. On the Raw Logs tab, choose **remote_addr > PV Distribution of IP Addresses and Request Method**.

   In the raw log entry, the request_method is GET and the status is 404.



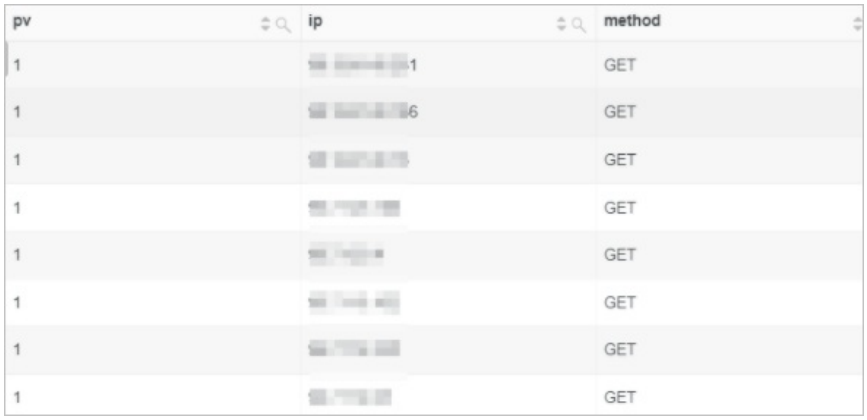
5. Click the name of the saved search. The following query statement is displayed in the window that appears:

```
* and request_method: GET and status: 400 | select count(*) as pv, remote_addr as ip,request_meth
od as method group by ip,method order by ip desc
```

6. View the query result of the saved search.

   In this example, the value of the static variable status2 is 400, which indicates the status field. The value of the request_method field is GET and the dynamic value of the variable method is GET. The result of the saved search shows the PV distribution of IP addresses whose request method is GET and status code is 400.

For example, the value of the request_method field is PUT. The result of the saved search shows the PV distribution of IP addresses whose request method is PUT and status code is 400.



pv	ip	method
1	[redacted]	GET
1	[redacted]	GET
1	[redacted]	GET
1	[redacted]	GET
1	[redacted]	GET
1	[redacted]	GET
1	[redacted]	GET
1	[redacted]	GET

8.Analysis grammar

8.1. SQL functions

8.1.1. Function overview

This topic describes the functions and operators that are involved in SQL analysis.

Aggregate functions

Function	Description
arbitrary function	Returns a random not-null value of the x field.
avg function	Returns the average of the values of the x field.
bitwise_and_agg function	Returns the result of the bitwise AND operation on the values of the x field.
bitwise_or_agg function	Returns the result of the bitwise OR operation on the values of the x field.
bool_and function	Checks whether all logs meet the specified condition. If all logs meet the specified condition, the function returns true. This function is equivalent to the every function.
bool_or function	Checks whether a log that meets the specified condition exists. If a log that meets the specified condition exists, the function returns true.
checksum function	Returns the checksum of the values of the x field.
count function	Returns the number of logs.
	Returns the number of logs. This function is equivalent to the count(*) function.
	Returns the number of logs whose value of the x field is not null.
count_if function	Returns the number of logs that meet the specified condition.
every function	Checks whether all logs meet the specified condition. If all logs meet the specified condition, the function returns true. This function is equivalent to the bool_and function.
geometric_mean function	Returns the geometric mean of the values of the x field.
kurtosis function	Returns the excess kurtosis of the values of the x field.
map_union function	Returns the union of the specified maps. If multiple input maps have the same key, the value of the key in the returned union is extracted from one of the maps at random.
max function	Returns the maximum value of the x field.
	Returns the n largest values of the x field. The function returns an array.
	Returns the value of the x field that corresponds to the maximum value of the y field.

Function	Description
max_by function	Returns the values of the x field that correspond to the n largest values of the y field. The function returns an array.
min function	Returns the minimum value of the x field.
	Returns the n smallest values of the x field. The function returns an array.
min_by function	Returns the value of the x field that corresponds to the minimum value of the y field.
	Returns the values of the x field that correspond to the n smallest values of the y field. The function returns an array.
skewness function	Returns the skewness of the values of the x field.
sum function	Returns the sum of the values of the x field.

String functions

Function	Description
chr function	Converts an ASCII code to characters.
codepoint function	Converts characters to an ASCII code.
concat function	Concatenates multiple strings into one string.
from_utf8 function	Decodes a binary string into a UTF-8 encoded string. Invalid UTF-8 sequences are replaced with the default replacement character U+FFFD.
	Decodes a binary string into a UTF-8 encoded string. Invalid UTF-8 sequences are replaced with a custom string.
length function	Returns the length of a string.
levenshtein_distance function	Returns the minimum edit distance between x and y.
lower function	Converts the characters in a string to lowercase letters.
lpad function	Left pads a string to a specified length by using a specified character and returns the result string.
ltrim function	Removes spaces from the start of a string.
normalize function	Normalizes a string to normalization form C (NFC).
position function	Returns the position of a specified substring in a string.
replace function	Replaces the matched characters in a string with specified characters.
	Removes the matched characters from a string.
reverse function	Returns a string in reverse order.
rpad function	Right pads a string to a specified length by using a specified character and returns the result string.

Function	Description
rtrim function	Removes spaces from the end of a string.
split function	Splits a string by using a specified delimiter and returns a set of substrings.
	Splits a string by using a specified delimiter and returns a set of substrings. The number of substrings that can be generated is specified by limit.
split_part function	Splits a string by using a specified delimiter and returns the substring at a specified position.
split_to_map function	Splits a string by using the first specified delimiter, and then splits the string by using the second specified delimiter.
strpos function	Returns the position of a specified substring in a string. This function is equivalent to the position(sub_string in x) function.
substr function	Returns the substring at a specified position in a string. The length of the substring is specified.
	Returns the substring at a specified position in a string.
to_utf8 function	Converts a string to a UTF-8 representation.
trim function	Removes spaces from the start and end of a string.
upper function	Converts the characters in a string to uppercase letters.

Date and time functions

Function	Description
current_date function	Returns the current date.
current_time function	Returns the current time and time zone.
current_timestamp function	Returns the current date, time, and time zone.
current_timezone function	Returns the current time zone.
date function	Returns the date part of a datetime expression.
date_format function	Converts a timestamp expression to a datetime expression in a specified format.
date_parse function	Converts a datetime string to a timestamp expression in a specified format.
from_iso8601_date function	Converts a date expression in the ISO 8601 format to a common date expression.
from_iso8601_timestamp function	Converts a datetime expression in the ISO 8601 format to a timestamp expression.
from_unixtime function	Converts a UNIX timestamp to a timestamp expression that does not contain a time zone.
	Converts a UNIX timestamp to a timestamp expression that contains a time zone.

Function	Description
	Converts a UNIX timestamp to a timestamp expression that contains a time zone. In the timestamp expression, the values for hours and minutes indicate the offset of the time zone.
localtime function	Returns the local time.
localtimestamp function	Returns the local date and time.
now function	Returns the current date and time. This function is equivalent to the current_timestamp function.
to_iso8601 function	Converts a date expression or a timestamp expression to a datetime expression in the ISO 8601 format.
to_unixtime function	Converts a timestamp expression to a UNIX timestamp.
day function	Returns the day of the month from a datetime expression. This function is equivalent to the day_of_month function.
day_of_month function	Returns the day of the month from a datetime expression. This function is equivalent to the day function.
day_of_week function	Returns the day of the week from a datetime expression. This function is equivalent to the dow function.
day_of_year function	Returns the day of the year from a datetime expression. This function is equivalent to the doy function.
dow function	Returns the day of the week from a datetime expression. This function is equivalent to the day_of_week function.
doy function	Returns the day of the year from a datetime expression. This function is equivalent to the day_of_year function.
extract function	Returns the specified field from a datetime expression. The field can be a date or a time.
hour function	Returns the hour of the day from a datetime expression. The 24-hour clock is used.
minute function	Returns the minute of the hour from a datetime expression.
month function	Returns the month of the year from a datetime expression.
quarter function	Returns the quarter of the year on which a specified date falls.
second function	Returns the second of the minute from a datetime expression.
timezone_hour function	Returns the offset of the time zone in hours.
timezone_minute function	Returns the offset of the time zone in minutes.

Function	Description
week function	Returns the week of the year on which a specified date falls. This function is equivalent to the week_of_year function.
week_of_year function	Returns the week of the year on which a specified date falls. This function is equivalent to the week function.
year function	Returns the year of a specified date.
year_of_week function	Returns the year of a specified date from a time expression that is specified in the ISO week date system. This function is equivalent to the yow function.
yow function	Returns the year of a specified date from a time expression that is specified in the ISO week date system. This function is equivalent to the year_of_week function.
date_trunc function	Truncates a datetime expression based on the time unit that you specify. The expression can be truncated based on the millisecond, second, minute, hour, day, month, or year.
date_add function	Adds N to the value of the x field based on the unit that you specify.
date_diff function	Returns the difference between two time expressions. For example, you can calculate the difference between x and y based on a specified time unit.
time_series function	Adds a value to the field that has no value returned in the specified time window.

JSON functions

Function	Description
json_array_contains function	Checks whether a JSON array contains a specified value.
json_array_get function	Returns the element that corresponds to an index in a JSON array.
json_array_length function	Returns the number of elements in a JSON array.
json_extract function	Returns a set of JSON values from a JSON object or a JSON array.
json_extract_scalar function	Returns a set of scalar values from a JSON object or a JSON array. The scalar values can be of the string, integer, or Boolean type. This function is similar to the json_extract function.
json_format function	Converts JSON data to a string.
json_parse function	Converts a string to JSON data.
json_size function	Returns the number of elements in a JSON object or a JSON array.

Regular expression functions

Function	Description
regexp_extract_all function	Extracts the substrings that match a specified regular expression from a specified string and returns an array of all matched substrings.
	Extracts the substrings that match a specified regular expression from a specified string and returns an array of substrings that match the nth capturing group in the regular expression.
regexp_extract function	Extracts the first substring that matches a specified regular expression from a specified string and returns the substring.
	Extracts the substrings that match a specified regular expression from a specified string and returns the first substring that matches the nth capturing group in the regular expression.
regexp_like function	Checks whether a specified string matches a specified regular expression.
regexp_replace function	Removes the substrings that match a specified regular expression from a specified string and returns the substrings that remain.
	Replaces the substrings that match a specified regular expression in a specified string and returns the result string.
regexp_split function	Splits a specified string into multiple substrings by using a specified regular expression and returns an array of the substrings.

Interval-valued comparison and periodicity-valued comparison functions

Function	Description
compare function	Compares the calculation result of the current time period with the calculation result of a time period n seconds before.
	Compares the calculation result of the current time period with the calculation results of time periods n1, n2, and n3 seconds before.
ts_compare function	Compares the calculation result of the current time period with the calculation result of a time period n seconds before. **Notice** The query and analysis results of the ts_compare function must be grouped by the time column by using the GROUP BY clause.
	Compares the calculation result of the current time period with the calculation results of time periods n1, n2, and n3 seconds before.

Array functions and operators

Function	Description
Subscript operator	Is used to retrieve the element at position x from an array.
array_agg function	Returns an array that is created from all values in x.
array_distinct function	Removes duplicate elements from an array.
array_except function	Returns the difference of two arrays.

Function	Description
array_intersect function	Returns the intersection of two arrays.
array_join function	Concatenates the elements of an array by using a specified delimiter and returns a string. If the array contains a null element, the null element is ignored.
	Concatenates the elements of an array by using a specified delimiter and returns a string. If the array contains a null element, the null element is replaced with null_replacement.
array_max function	Returns the maximum value in an array.
array_min function	Returns the minimum value in an array.
array_position function	Returns the position of a specified element in an array. Positions start from 1. If the specified element does not exist, the function returns 0.
array_remove function	Removes a specified element from an array.
array_sort function	Returns an array of elements that are sorted in ascending order. If the array contains a null element, the null element is placed at the end.
array_transpose function	Transposes a matrix and returns a new two-dimensional array that is created from the elements in the matrix. The elements are located by using the same indexes.
array_union function	Returns the union of two arrays.
cardinality function	Returns the number of elements in an array.
concat function	Returns the concatenation of multiple arrays.
contains function	Checks whether an array contains a specified element. If the array contains the specified element, the function returns true.
element_at function	Returns the element at position y in an array.
filter function	Filters elements in an array based on a lambda expression and returns elements that meet the expression.
flatten function	Transforms a two-dimensional array into a one-dimensional array.
reduce function	Sums each element in an array by using a lambda expression and returns the result.
reverse function	Returns an array of elements that are sorted in reverse order.
sequence function	Returns an array of elements within a specified range. The elements are consecutive and incremental. The incremental step is 1, which is the default value.
	Returns an array of elements within a specified range. The elements are incremental. The incremental step is a custom value.
shuffle function	Returns a random permutation of elements in an array.
slice function	Returns a subset of an array.

Function	Description
transform function	Transforms each element in an array by using a lambda expression.
zip function	Merges multiple arrays into a two-dimensional array. Elements that have the same position in the input arrays form a new array in the two-dimensional array.
zip_with function	Merges two arrays into a single array by using a lambda expression.

Map functions and operators

Function	Description
Subscript operator	Is used to retrieve the value of a key from a map.
cardinality function	Returns the size of a map.
element_at function	Returns the value of a key in a map.
histogram function	Groups query and analysis results and returns data in the JSON format.
histogram_u function	Groups query and analysis results and returns data in multiple rows and multiple columns.
map function	Returns an empty map.
	Returns a map that is created by using two arrays.
map_agg function	Returns a map that is created by using x and y. x is the key in the map. y is the value of the key in the map. If y has multiple values, a random value is extracted as the value of the key.
map_concat function	Returns the union of multiple maps.
map_filter function	Filters elements in a map based on a lambda expression and returns a new map.
map_keys function	Returns an array that consists of all the keys of a map.
map_values function	Returns an array that consists of all the values of a map.
multimap_agg function	Returns a multimap that is created by using x and y. x is the key in the multimap. y is the value of the key in the multimap, and the value is of the array type. If y has multiple values, all the values are extracted as the values of the key.

Mathematical calculation functions

Function	Description
abs function	Calculates the absolute value of x.
acos function	Calculates the arc cosine of x.
asin function	Calculates the arc sine of x.
atan function	Calculates the arc tangent of x.

Function	Description
atan2 function	Calculates the arc tangent of x divided by y.
cbrt function	Calculates the cube root of x.
ceil function	Rounds x up to the nearest integer. This function is an alias of the ceiling function.
ceiling function	Rounds x up to the nearest integer.
cos function	Calculates the cosine of x.
cosh function	Calculates the hyperbolic cosine of x.
cosine_similarity function	Calculates the cosine similarity between x and y.
degrees function	Converts an angle in radians to its equivalent in degrees.
e function	Returns the value of e, which is the base of the natural logarithm.
exp function	Raises e to the power of x.
floor function	Rounds x down to the nearest integer.
from_base function	Converts x to a base-y number.
ln function	Calculates the natural logarithm of x.
infinity function	Returns a value that represents positive infinity.
is_nan function	Determines whether x is Not a Number (NaN).
log2 function	Calculates the base-2 logarithm of x.
log10 function	Calculates the base-10 logarithm of x.
log function	Calculates the base-y logarithm of x.
mod function	Calculates the remainder of x divided by y.
nan function	Returns a value that is NaN.
pi function	Returns the value of π to 15 decimal places.
pow function	Raises x to the power of y. This function is an alias of the power function.
power function	Raises x to the power of y.
radians function	Converts an angle in degrees to its equivalent in radians.
rand function	Returns a random number.
random function	Returns a random number in the range [0,1). Returns a random number in the range [0,x).

Function	Description
round function	Rounds x to the nearest integer.
	Rounds x to the nearest decimal with n decimal places.
sign function	Returns the sign of x. Valid values: 1, 0, and -1.
sin function	Calculates the sine of x.
sqrt function	Calculates the square root of x.
tan function	Calculates the tangent of x.
tanh function	Calculates the hyperbolic tangent of x.
to_base function	Converts x to a base-y string.
truncate function	Removes the fractional part of x.
width_bucket function	Divides a numeric range into buckets of equal width and returns the bucket number of x.
	Returns the bucket number of x in the range of buckets that are specified by an array.

Mathematical statistics functions

Function	Description
corr function	Returns the coefficient of correlation between x and y. The return value is in the range of [0,1].
covar_pop function	Returns the population covariance of x and y.
covar_samp function	Returns the sample covariance of x and y.
regr_intercept function	Returns the y-intercept of the line for the linear equation that is determined by the `(x,y)` pair.
regr_slope function	Returns the slope of the line for the linear equation that is determined by the `(x,y)` pair.
stddev function	Returns the sample standard deviation of x. This function is equivalent to the stddev_samp function.
stddev_samp function	Returns the sample standard deviation of x.
stddev_pop function	Returns the population standard deviation of x.
variance function	Returns the sample variance of x. This function is equivalent to the var_samp function.
var_samp function	Returns the sample variance of x.
var_pop function	Returns the population variance of x.

Data type conversion functions

Function	Description
cast function	Converts x from one data type to another. If the cast function fails to convert a value, the query that calls this function is terminated.
try_cast function	Converts x from one data type to another. If the try_cast function fails to convert a value, the function returns NULL. The query that calls this function can process the NULL value and continue to run. **Note** A log may contain dirty data. When you query logs, we recommend that you use the try_cast function. This way, conversion failures do not cause your queries to fail.
typeof function	Returns the data type of x.

Security check functions

Function	Description
security_check_ip function	Checks whether an IP address is secure.
security_check_domain function	Checks whether a domain name is secure.
security_check_url function	Checks whether a URL is secure.

Window functions

Function	Description
Aggregate functions	You can use all aggregate functions as window functions. For more information about aggregate functions, see Aggregate function.
cume_dist function	Calculates the cumulative distribution of each value in a partition. The result is obtained by using division. The numerator is the number of rows whose field values are smaller than or equal to the field value of the specified row. The specified row is also counted. The denominator is the total number of rows in the partition. The calculation is based on the order of the rows in the partition. The return value is in the range of (0,1].
dense_rank function	Calculates the rank of each value of a specified field in a partition. If two values are the same, they are assigned the same rank. The ranks are consecutive. For example, if two values are assigned the same rank of 1, the next rank is 2.
ntile function	Divides the rows in each partition into n groups based on a specified order.
percent_rank function	Calculates the percentage ranking of each row in a partition.
rank function	Calculates the rank of each value of a specified field in a partition. If two values are the same, they are assigned the same rank. The ranks are not consecutive. For example, if two values are assigned the same rank of 1, the next rank is 3.

Function	Description
row_number function	Calculates the rank of each value of a specified field in a partition. Each value is assigned a unique rank. The ranks start from 1. For example, if three values are the same, they are assigned the ranks of 1, 2, and 3.
first_value function	Returns the value of a specified field in the first row of each partition.
last_value function	Returns the value of a specified field in the last row of each partition.
lag function	Returns the value of a specified field in the row that is at the specified offset before the current row in a partition. If no row exists at the specified offset before the current row, the value that is specified by default_value is returned.
lead function	Returns the value of a specified field in the row that is at the specified offset after the current row in a partition. If no row exists at the specified offset after the current row, the value that is specified by default_value is returned.
nth_value function	Returns the value of a specified field in the row that is at the specified offset from the beginning of a partition.

IP functions

Function	Description
ip_to_city function	Identifies the city to which an IP address belongs. The function returns the Chinese name of a city.
	Identifies the city to which an IP address belongs. The function returns the administrative region code of a city.
ip_to_city_geo function	Identifies the longitude and latitude of the city to which an IP address belongs. The function returns the longitude and latitude of a city. Each city has only one set of coordinates.
ip_to_country function	Identifies the country or region to which an IP address belongs. The function returns the Chinese name of a country or region.
	Identifies the country or region to which an IP address belongs. The function returns the code of a country or region.
ip_to_country_code function	Identifies the country or region to which an IP address belongs. The function returns the code of a country or region.
ip_to_domain function	Checks whether an IP address is a private or a public address.
ip_to_geo function	Identifies the longitude and latitude of the location to which an IP address belongs.
ip_to_provider function	Identifies the Internet service provider (ISP) of an IP address.
	Identifies the state to which an IP address belongs. The function returns the Chinese name of a state.

Function	Description
ip_to_province function	Identifies the state to which an IP address belongs. The function returns the administrative region code of a state.
ip_prefix function	Returns the prefix of an IP address.
is_prefix_subnet_of function	Checks whether a CIDR block is a subnet of a specified CIDR block.
is_subnet_of function	Checks whether an IP address is in a specified CIDR block.
ip_subnet_max function	Returns the largest IP address in a CIDR block.
ip_subnet_min function	Returns the smallest IP address in a CIDR block.
ip_subnet_range function	Returns the range of a CIDR block.

URL functions

Function	Description
url_encode function	Encodes a URL.
url_decode function	Decodes a URL.
url_extract_fragment function	Extracts the fragment from a URL.
url_extract_host function	Extracts the host from a URL.
url_extract_parameter function	Extracts the value of a specified parameter in the query string from a URL.
url_extract_path function	Extracts the path from a URL.
url_extract_port function	Extracts the port number from a URL.
url_extract_protocol function	Extracts the protocol from a URL.
url_extract_query function	Extracts the query string from a URL.

Approximate functions

Function	Description
approx_distinct function	Estimates the number of distinct values in the x field. The standard error is 2.3%, which is the default value.
	Estimates the number of distinct values in the x field. The standard error is a custom value.
approx_percentile function	Returns the approximate histogram of x based on the number of histogram columns that are specified by the bucket parameter. The returned result is of the JSON type.
	Returns the approximate histogram of x based on the number of histogram columns that are specified by the bucket parameter. The returned result is of the JSON type. You can specify a weight for x.

Function	Description
numeric_histogram function	Returns the approximate histogram of x based on the number of histogram columns that are specified by the bucket parameter. The returned result contains multiple rows and columns.
approx_percentile function	Sorts the values of x in ascending order and returns the value that is approximately at the specified percentile.
	Sorts the values of x in ascending order and returns the values that are approximately at percentile01 and percentile02.
	Sorts the values of x in ascending order based on the products of the values of x and the weight and returns the value of x that is approximately at the specified percentile.
	Sorts the values of x in ascending order based on the products of the values of x and the weight and returns the value of x that are approximately at percentile01 and percentile02.
	Sorts the values of x in ascending order based on the products of the values of x and the weight and returns the value of x that is approximately at the specified percentile. You can configure the accuracy of the returned results.

Binary functions

Function	Description
from_base64 function	Decodes a Base64-encoded string into a binary number.
from_base64url function	Decodes a Base64-encoded string into a binary number by using URL reserved characters.
from_big_endian_64 function	Converts a binary number in big endian to a bigint value.
from_hex function	Converts a hexadecimal number to a binary number.
length function	Returns the length of a binary number.
md5 function	Computes the MD5 hash value for a binary number.
to_base64 function	Encodes a binary number into a Base64 string representation.
to_base64url function	Encodes a binary number into a Base64 string representation by using URL reserved characters.
to_hex function	Converts a binary number to a hexadecimal number.
to_big_endian_64 function	Encodes a bigint value into a binary number in big endian.
sha1 function	Computes the SHA-1 hash value for a binary number.
sha256 function	Computes the SHA-256 hash value for a binary number.
sha512 function	Computes the SHA-512 hash value for a binary number.
xxhash64 function	Computes the xxhash64 hash value for a binary number.

Bitwise functions

Function	Description
bit_count function	Returns the number of bits 1 in x in binary representation.
bitwise_and function	Returns the result of the bitwise AND operation on x and y in binary representation.
bitwise_not function	Returns the result of the bitwise NOT operation on x in binary representation.
bitwise_or function	Returns the result of the bitwise OR operation on x and y in binary representation.
bitwise_xor function	Returns the result of the bitwise XOR operation on x and y in binary representation.

Geospatial functions

Function	Description
ST_AsText function	Returns the well-known text (WKT) representation of a geometry.
ST_GeometryFromText function	Returns a geometry from the specified WKT representation.
ST_LineFromText function	Returns a line string from the specified WKT representation.
ST_Polygon function	Returns a polygon from the specified WKT representation.
ST_Point function	Returns a point from the specified WKT representation.
ST_Boundary function	Returns the closure of the combinatorial boundary of a geometry.
ST_Buffer function	Returns a geometry that represents all points whose distance from the specified geometry is less than or equal to the specified distance.
ST_Difference function	Returns a geometry that represents the point set difference of two specified geometries.
ST_Envelope function	Returns the bounding rectangular polygon of a geometry.
ST_ExteriorRing function	Returns a line string that represents the exterior ring of a geometry.
ST_Intersection function	Returns a geometry that represents the point set intersection of two specified geometries.
ST_SymDifference function	Returns a geometry that represents the point set symmetric difference of two specified geometries.
ST_Contains function	Returns true if no points of the second geometry lie in the exterior of the first geometry and at least one point of the interior of the first geometry lies in the interior of the second geometry.
ST_Crosses function	Returns true if two specified geometries have several interior points in common.
ST_Disjoint function	Returns true if two specified geometries do not share any portion of the two-dimensional space.
ST_Equals function	Returns true if two specified geometries represent the same geometry.

Function	Description
ST_Intersects function	Returns true if two specified geometries share a portion of the two-dimensional space.
ST_Overlaps function	Returns true if two specified geometries share space and have the same dimension but are not completely contained by each other.
ST_Relate function	Returns true if two specified geometries have a spatial relationship.
ST_Touches function	Returns true if two specified geometries have at least one point in common but their interiors do not intersect.
ST_Within function	Returns true if the first geometry is completely inside the second geometry.
ST_Area function	Calculates the projected area of a geometry on a two-dimensional plane by using the Euclidean distance method.
ST_Centroid function	Returns the point value that represents the mathematical centroid of a geometry.
ST_CoordDim function	Returns the coordinate dimension of a geometry.
ST_Dimension function	Returns the inherent dimension of a geometry. The inherent dimension must be less than or equal to the coordinate dimension.
ST_Distance function	Returns the minimum distance between two geometries.
ST_EndPoint function	Returns the last point of a line string.
ST_IsClosed function	Returns true if the start point of a line string coincides with the end point.
ST_IsEmpty function	Returns true if a geometry is empty.
ST_IsRing function	Returns true if a line string is closed and simple.
ST_Length function	Calculates the projected length of a line string on a two-dimensional plane by using the Euclidean distance method. If multiple line strings exist, the function returns the sum of the lengths of the multiple line strings.
ST_NumPoints function	Returns the number of points in a geometry.
ST_NumInteriorRing function	Returns the number of interior rings in a geometry.
ST_StartPoint function	Returns the first point of a line string.
ST_X function	Returns the X coordinate of a specified point.
ST_XMax function	Returns the maximum first X coordinate of a geometry.
ST_XMin function	Returns the minimum first X coordinate of a geometry.
ST_Y function	Returns the Y coordinate of a specified point.
ST_YMax function	Returns the maximum first Y coordinate of a geometry.
ST_YMin function	Returns the minimum first Y coordinate of a geometry.
	Returns a Bing tile based on the X coordinate, Y coordinate, and zoom level.

Function	Description
bing_tile function	Returns a Bing tile based on the quadtree key.
bing_tile_at function	Returns a Bing tile based on the latitude, longitude, and zoom level.
bing_tile_coordinates function	Returns the X and Y coordinates of a Bing tile.
bing_tile_polygon function	Returns the polygon format of a Bing tile.
bing_tile_quadkey function	Returns the quadtree key of a Bing tile.
bing_tile_zoom_level function	Returns the zoom level of a Bing tile.

Geo functions

Function	Description
geohash function	Encodes latitudes and longitudes by using the Geohash algorithm.

Color functions

Function	Description
bar function	Returns a part of an ANSI bar chart. You can configure the width parameter to specify the width of the ANSI bar chart. However, you cannot configure the high_color and low_color parameter to specify the colors for the chart. The default values of the high_color and low_color parameters are used. The default value of the low_color parameter is red and the default value of the high_color parameter is green. In addition, you can configure the x parameter to specify the length of the part that is returned by the function.
	Returns a part of an ANSI bar chart. You can configure the width parameter to specify the width of the ANSI bar chart. You can also configure the high_color and low_color parameters to specify the custom colors for the chart. In addition, you can configure the x parameter to specify the length of the part that is returned by the function.
color function	Converts a color string to a color type.
	Returns a color between high_color and low_color based on the portions of high_color and low_color. The portions are determined by the proportion of x between high and low.
	Returns a color between high_color and low_color based on the portions of high_color and low_color. The portions are determined by y.
render function	Returns results by using color rendering. If the Boolean expression evaluates to true, the function returns a green tick. If the Boolean expression evaluates to false, the function returns a red cross.
	Returns results by using custom color rendering.
rgb function	Returns a color value based on an RGB value.

HyperLogLog functions

Function	Description
approx_set function	Estimates the number of distinct values in the x field. The maximum standard error is 0.01625, which is the default value.
cardinality function	Converts HyperLogLog data to bigint data.
empty_approx_set function	Returns a null value of the HyperLogLog type. The maximum standard error is 0.01625, which is the default value.
merge function	Aggregates all HyperLogLog values.

Comparison operators

Operator	Description
Relational operators	Is used to compare x with y. If the specified condition is met, true is returned.
all operator	If x meets all conditions, true is returned.
any operator	If x meets one of the conditions, true is returned.
between operator	If x is between y and z, true is returned.
distinct operator	If x is not equal to y, true is returned.
	If x is equal to y, true is returned.
like operator	Is used to match a specified character pattern in a string. The string is case-sensitive.
some operator	If x meets one of the conditions, true is returned.
greatest operator	Is used to obtain the greater value of x and y.
least operator	Is used to obtain the smaller value of x and y.
null operator	If x is null, true is returned.
	If x is not null, true is returned.

Logical operators

Operator	Description
AND operator	If both x and y evaluate to true, true is returned.
OR operator	If either x or y evaluates to true, true is returned.
NOT operator	If x evaluates to false, true is returned.

Unit conversion functions

Function	Description

Function	Description
convert_data_size function	Converts a measurement from the current unit to the optimal unit. The system automatically determines the optimal unit and returns a measurement in the optimal unit. The returned result is of the string type. For example, you can convert 1,024 KB to 1 MB and 1,024 MB to 1 GB.
	Converts a measurement from the current unit to a specified unit. The returned result is of the string type.
format_data_size function	Converts a measurement in byte to a measurement in a specified unit. The returned result is of the string type.
parse_data_size function	Converts a measurement from the current unit to a measurement in byte. The returned result is of the decimal type.
to_data_size_B function	Converts a measurement from the current unit to a measurement in byte. The returned result is of the double type.
to_data_size_KB function	Converts a measurement from the current unit to a measurement in KB. The returned result is of the double type.
to_data_size_MB function	Converts a measurement from the current unit to a measurement in MB. The returned result is of the double type.
to_data_size_GB function	Converts a measurement from the current unit to a measurement in GB. The returned result is of the double type.
to_data_size_TB function	Converts a measurement from the current unit to a measurement in TB. The returned result is of the double type.
to_data_size_PB function	Converts a measurement from the current unit to a measurement in PB. The returned result is of the double type.
format_duration function	Converts a time interval in seconds to a readable string.
parse_duration function	Converts a time interval to a time interval in the `0 00:00:00.000` format.
to_days function	Converts a time interval to a time interval in days.
to_hours function	Converts a time interval to a time interval in hours.
to_microseconds function	Converts a time interval to a time interval in microseconds.
to_milliseconds function	Converts a time interval to a time interval in milliseconds.
to_minutes function	Converts a time interval to a time interval in minutes.
to_most_succinct_time_unit function	Converts a time interval from the current unit to the optimal unit. The system automatically determines the optimal unit and returns a time interval in the optimal unit.
to_nanoseconds function	Converts a time interval to a time interval in nanoseconds.
to_seconds function	Converts a time interval to a time interval in seconds.

Window funnel functions

Function	Description
window_funnel function	Searches the event chain in the sliding time window and counts the maximum number of consecutive events that occurred in the event chain.

Lambda expressions

Log Service allows you to define a lambda expression in an analytic statement and pass the expression to a specified function. For more information, see Lambda expressions.

Conditional expressions

Expression	Description
CASE WHEN statement	Categorizes data by using conditions.
IF function	Categorizes data by using conditions.
COALESCE function	Returns the first not-null value based on multiple expressions.
NULLIF function	Compares the values of two expressions. If the values of the two expressions are equal, null is returned. Otherwise, the value of the first expression is returned.
TRY function	Evaluates an expression to capture errors. This helps the system continue to query and analyze data.

8.1.2. Aggregate function

An aggregate function calculates the values of a field and returns a single value. This topic describes the syntax of aggregate functions. This topic also provides examples on how to use aggregate functions.

The following table describes the aggregate functions that are supported by Log Service.

Notice

Function	Syntax	Description
arbitrary function	arbitrary()	Returns a random, non-null value of the field.
avg function	avg()	Calculates the average of the values of the field.
bitwise_and_agg function	bitwise_and_agg()	Returns the result of the bitwise AND operation on the values of the field.
bitwise_or_agg function	bitwise_or_agg()	Returns the result of the bitwise OR operation on the values of the field.
bool_and function	bool_and(*boolean expression*)	Checks whether all log entries meet the specified condition. If all log entries meet the specified condition, the function returns true. This function is equivalent to the every function.
bool_or function	bool_or(*boolean expression*)	Checks whether a log entry that meets the specified condition exists. If a log entry that meets the specified condition exists, the function returns true.

Function	Syntax	Description
checksum function	checksum()	Calculates the checksum of the values of the field.
count function	count(*)	Counts the number of log entries.
	count(1)	Counts the number of log entries. This function is equivalent to the count(*) function.
	count()	Counts the number of log entries that contain the field whose value is not null.
count_if function	count_if(*boolean expression*)	Counts the number of log entries that meet the specified condition.
every function	every(*boolean expression*)	Checks whether all log entries meet the specified condition. If all log entries meet the specified condition, the function returns true. This function is equivalent to the bool_and function.
geometric_mean function	geometric_mean()	Calculates the geometric mean of the values of the field.
kurtosis function	kurtosis()	Calculates the excess kurtosis of the values of the field.
map_union function	map_union()	Returns the result of the union operation on the specified maps. If multiple input maps have the same key, the function selects an arbitrary input map from these input maps to be in the output map.
max function	max()	Queries the largest value of the field.
	max(,*n*)	Queries the *n* largest values of the field. The function returns an array.
max_by function	max_by(,)	Queries the value of that is associated with the largest value of the field.
	max_by(,,*n*)	Queries the values of that are associated with the *n* largest values of the field.
min function	min()	Queries the minimum value of the field.
	min(,*n*)	Queries the *n* smallest values of the field. The function returns an array.
min_by function	min_by(,)	Queries the value of that is associated with the smallest value of the field.
	min_by(,,*n*)	Queries the values of that are associated with the *n* smallest values of the field. The function returns an array.
skewness function	skewness()	Calculates the skewness of the values of the field.
sum function	sum()	Calculates the sum of the values of the field.

arbitrary function

Returns an arbitrary, non-null value of the field.

```
arbitrary()
```

Parameter	Description
	The value of this parameter can be of any data type.

The data type of the return value is the same as the data type of the parameter.

Returns an arbitrary, non-null value of the request_method field.

- Query statement

```
* | SELECT arbitrary(request_method) AS request_method
```

- Query result

request_method
GET

avg function

Calculates the average of the values of the field.

```
avg()
```

Parameter	Description
	The parameter can be of the double, bigint, decimal, or real data type.

Data of the double data type.

Return the projects whose average latency of a defined item is greater than 1,000 microseconds.

- Query statement

```
method: PostLogstoreLogs | SELECT avg(latency) AS avg_latency, Project GROUP BY Project HAVING avg
_latency > 1000
```

- Query result

avg_latency	Project
3223.4162679425835	datalab-14[redacted]61-cn-chengdu

bitwise_and_agg function

Returns the result of the bitwise AND operation on the values of the field.

```
bitwise_and_agg()
```

Parameter	Description
	The parameter can be of the bigint data type.

Data of the bigint data type. The result represents a binary.

Returns the result of the bitwise AND operation on the values of the request_time field.

- Query statement

```
* | SELECT bitwise_and_agg(status)
```

- Query result

_col0
0

bitwise_or_agg function

Returns the result of the bitwise OR operation on the values of the field.

```
bitwise_or_agg()
```

Parameter	Description
	The parameter can be of the bigint data type.

Data of the bigint data type. The result represents a binary.

Returns the result of the bitwise OR operation on the values of the request_time field.

- Query statement

```
* | SELECT bitwise_or_agg(request_length)
```

- Query result

_col0
16383

bool_and function

Checks whether all log entries meet the specified condition. If all log entries meet the specified condition, the function returns true. This function is equivalent to the every function.

```
bool_and(boolean expression)
```

Parameter	Description
boolean expression	The parameter can be a boolean expression.

Data of the boolean data type.

Checks whether the value of the request_time field is less than 100 in all log entries. Unit: seconds. If the value of the request_time field is less than 100 in all log entries, the function returns true.

- Query statement

```
* | SELECT bool_and(request_time < 100)
```

- Query result

_col0
true

bool_or function

Checks whether a log entry that meets the specified condition exists. If a log entry that meets the specified condition exists, the function returns true.

```
bool_or(boolean expression)
```

Parameter	Description
boolean expression	The parameter can be a boolean expression.

Data of the boolean type.

Checks whether a log entry in which the value of the request_time field is less than 20 exists. Unit: seconds. If a log entry in which the value of the request_time field is less than 20 exists, the function returns true.

- Query statement

```
* | SELECT bool_or(request_time < 20)
```

- Query result

checksum function

Calculates the checksum of the values of the field.

```
checksum()
```

Parameter	Description
	The value of this parameter can be of any data type.

Data of the string data type. The result is Base64-encoded.

- Query statement

```
* | SELECT checksum(request_method) AS request_method
```

- Query result

count function

Counts the number of log entries.

- Counts the number of log entries.

```
count(*)
```

- Counts the number of log entries. This function is equivalent to the `count(*)` function.

```
count(1)
```

- Counts the number of log entries that contain the field whose value is not null.

```
count()
```

Parameter	Description
	The value of this parameter can be of any data type.

Data of the integer data type.

- Example 1: Counts the page view (PV) of a website.
  - Query statement

```
* | SELECT count(*) AS PV
```

  - Query result

PV
2009

- Example 2: Counts the number of log entries that have the request_method field whose value is not null.
  - Query statement

```
* | SELECT count(request_method) AS count
```

  - Query result

count
1954

count_if function

Counts the number of log entries that meet the specified condition.

```
count_if(boolean expression)
```

Parameter	Description
boolean expression	The parameter can be a boolean expression.

Data of the integer data type.

Count the log entries that have the request_uri field whose value ends with `file-0` .

- Query statement

```
* | SELECT count_if(request_uri like '%file-0') AS count
```

- Query result

count
1954

geometric_mean function

Calculates the geometric mean of the values of the field.

```
geometric_mean()
```

Parameter	Description
	The parameter can be of the double, bigint, or real data type.

Data of the double data type.

Calculates the geometric mean of request times.

- Query statement

```
* | SELECT geometric_mean(request_time) AS time
```

- Query result

time
39.443123208882308

every function

Checks whether all log entries meet the specified condition. If all log entries meet the specified condition, the function returns true. This function is equivalent to the bool_and function.

```
every(boolean expression)
```

Parameter	Description
boolean expression	The parameter can be a boolean expression.

Data of the boolean data type.

Checks whether all request times are less than 100 seconds. If all request times are less than 100 seconds, the function returns true.

- Query statement

```
* | SELECT every(request_time < 100)
```

- Query result

_col0
true

kurtosis function

Calculates the excess kurtosis of the values of the field.

```
kurtosis()
```

Parameter	Description
	The parameter can be of the double or bigint data type.

Data of the double data type.

Calculates the excess kurtosis of a set of request time.

- Query statement

```
*| SELECT kurtosis(request_time)
```

- Query result

map_union function

Returns the result of the union operation on the maps that parameter specifies. If multiple maps have the same key, the function selects an arbitrary input map from these input maps to be in the output map.

```
map_union()
```

Parameter	Description
	The parameter can be of the map data type.

Data of the map data type.

Perform a union operation on the maps of the etl_context field and returns a map. If multiple input maps have the same key, the function selects an arbitrary input map from these input maps to be in the output map.

- Field example

```
etl_context: {
 project:"datalab-148****6461-cn-chengdu"
 logstore:"internal-etl-log"
 consumer_group:"etl-83****4d1965"
 consumer:"etl-b2d40ed****c8d6-291294"
 shard_id:"0" }
```

- Query statement

```
* | SELECT map_union(try_cast(json_parse(etl_context) AS map(varchar,varchar)))
```

- Query result

max function

Queries the largest value of the field.

- Returns the largest value.

```
max()
```

- Returns the n largest values.

```
max(,n)
```

Parameter	Description
	The value of this parameter can be of any data type.
n	The value must be a positive integer.

The data type of the return value is the same as the data type of the parameter.

- Example 1: Queries the longest request duration.
  - Query statement

```
* | SELECT max(request_time) AS max_request_time
```

  - Query result

- Example 2: Queries the 10 longest request durations.
  - Query statement

```
* | SELECT max(request_time,10) AS "top 10"
```

  - Query result

max_by function

The following list shows the syntax that is supported by the max_by function.

- Queries the value of the field that is associated with the largest value of the field.

```
max_by(,)
```

- Queries the values of the field that is associated with the *n* largest values of the field.

```
max_by(,,n)
```

Parameter	Description
	The value of this parameter can be of any data type.
	The value of this parameter can be of any data type.
n	The value must be a positive integer.

The data type of the return value is the same as the data type of the parameter.

- Example 1: Queries the time of the order that has the highest value.
  - Query statement

```
* | SELECT max_by(UsageEndTime, PretaxAmount) as time
```

- Query result

time
1625731025

- Example 2: Queries the request methods of the 3 requests that have the longest request durations.
  - Query statement

```
* | SELECT max_by(request_method,request_time,3) AS method
```

  - Query result

method
["POST","POST","POST"]

min function

Queries the minimum value of the field.

- Returns the minimum value.

```
min()
```

- Returns the n minimum values. The function returns an array.

```
min(,n)
```

Parameter	Description
	The value of this parameter can be of any data type.
n	The value must be a positive integer.

The data type of the return value is the same as the data type of the parameter.

- Example 1: Queries the shortest request duration.
  - Example

```
* | SELECT min(request_time) AS min_request_time
```

  - Query result

min_request_time
10.0

- Example: Queries the 10 shortest request durations.
  - Query statement

```
* | SELECT min(request_time,10)
```

  - Query result

_col0
[10.0,10.0,10.0,10.0,10.0,11.0,12.0,12.0,13.0,13.0]

min_by function

The following list shows the syntax that is supported by the min_by function.

- Queries the value of the field that is associated with the smallest value of the field.

```
min_by(,)
```

- Queries the values of the field that is associated with the *n* smallest values of the field. The function returns an array.

```
min_by(,,n)
```

Parameter	Description
	The value of this parameter can be of any data type.
	The value of this parameter can be of any data type.
n	The value must be a positive integer.

The data type of the return value is the same as the data type of the parameter.

- The following query statement returns the request method of the request that has the shortest request duration:
  - Query statement

```
* | SELECT min_by(request_method,request_time) AS method
```

  - Query result

- The following query statement returns the request methods of the requests whose request durations are the three shortest request durations:
  - Query statement

```
* | SELECT min_by(request_method,request_time,3) AS method
```

  - Query result

skewness function

Calculates the skewness of the values of the field.

```
skewness()
```

Parameter	Description
	The parameter can be of the double or bigint data type.

Data of the double data type.

Calculates the skewness of the request times.

- Query statement

```
*| SELECT skewness(request_time) AS skewness
```

- Query result

skewness
0.0004294441602043965

sum function

Calculates the sum of the values of field.

```
sum()
```

Parameter	Description
	The parameter can be of the double, bigint, decimal, or real data type.

The data type of the return value is the same as the data type of the parameter.

Calculates the daily inbound traffic of the website.

- Query statement

```
* | SELECT date_trunc('day',__time__) AS time, sum(body_bytes_sent) AS body_bytes_sent GROUP BY time ORDER BY time
```

- Query result

time	body_bytes_sent
2021-07-06 00:00:00.000	3925728

8.1.3. String functions

This topic describes the syntax of string functions. This topic also provides examples on how to use the functions.

The following table describes the string functions that are supported by Log Service.

Notice

Function	Syntax	Description
chr function	chr()	Converts an ASCII code to characters.
codepoint function	codepoint()	Converts characters to an ASCII code.
concat function	concat(,...)	Concatenates multiple strings into one string.
from_utf8 function	from_utf8()	Decodes a binary string into a UTF-8-encoded string. Invalid UTF-8 sequences are replaced with the default replacement character U+FFFD.

Function	Syntax	Description
from_utf8 function	from_utf8(,*replace_string*)	Decodes a binary string into a UTF-8 encoded string. Invalid UTF-8 sequences are replaced with a custom string.
length function	length()	Returns the length of a string.
levenshtein_distance function	levenshtein_distance(,)	Returns the minimum edit distance between and .
lower function	lower()	Converts the characters in a string to lowercase letters.
lpad function	lpad(,*length*,*lpad_string*)	Left pads a string to a specified length by using a specified character and returns the result string.
ltrim function	ltrim()	Removes space from the start of a string.
normalize function	normalize()	Transforms a string by using the NFC normalization form.
position function	position(*sub_string* in)	Returns the position of a specified substring in a string.
replace function	replace(,*sub_string*,*replace_string*)	Replaces the matched characters in a string with specified characters.
	replace(,*sub_string*)	Removes the matched characters from a string.
reverse function	reverse()	Returns a string in reverse order.
rpad function	rpad(,*length*,*rpad_string*)	Right pads a string to a specified length by using a specified character and returns the result string.
rtrim function	rtrim()	Removes spaces from the end of a string.
split function	split(,*delimeter*)	Splits a string by using a specified delimiter and returns a set of substrings.
	split(,*delimeter*,*limit*)	Splits a string by using a specified delimiter and returns a set of substrings. The number of substrings that can be generated is specified by *limit*.
split_part function	split_part(,*delimeter*,*part*)	Splits a string by using a specified delimiter and returns the substring at a specified position.
split_to_map function	split_to_map(,*delimiter01*,*delimiter02*)	Splits a string by using the first specified delimiter, and then splits the string by using the second specified delimiter.
strpos function	strpos(,*sub_string*)	Returns the position of a specified substring in a string. This function is equivalent to the position(*sub_string* in) function.

Function	Syntax	Description
substr function	substr(,*start*)	Returns the substring at a specified position in a string.
	substr(,*start*,*length*)	Returns the substring at a specified position in a string. The length of the substring is specified.
to_utf8 function	to_utf8()	Converts a string to a UTF-8 representation.
trim function	trim()	Removes spaces from the start and end of a string.
upper function	upper()	Converts the characters in a string to uppercase letters.

chr function

The chr function converts an ASCII code to characters.

```
chr()
```

Parameter	Description
	The value of this parameter is an ASCII code.

The varchar type.

Check whether the first letter in the value of the region field starts with the letter c. In the following query statement, the value 99 is an ASCII code and represents the lowercase letter c.

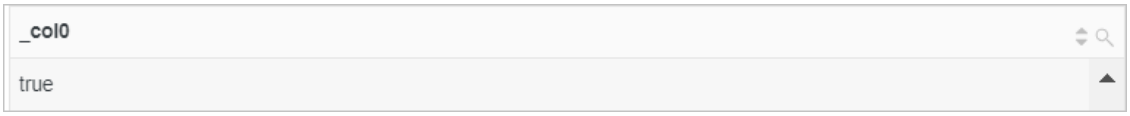
- Sample field

  ```
  region:cn-shanghai
  ```

- Query statement

  ```
  * | SELECT substr(region, 1, 1)=chr(99)
  ```

- Query and analysis results

codepoint function

The codepoint function converts characters to an ASCII code.

```
codepoint()
```

Parameter	Description
	The value of this parameter is of the varchar type.

The integer type.

Check whether the first letter in the value of the region field starts with the letter c. In the following query statement, the value 99 is an ASCII code and represents the lowercase letter c.

- Sample field

```
upstream_status:200
```

- Query statement

```
* | SELECT codepoint(cast (substr(region, 1, 1) AS char(1))) =99
```

- Query and analysis results

concat function

The concat function concatenates multiple strings into one string.

```
concat(,...)
```

Parameter	Description
	The value of this parameter is of the varchar type.
	The value of this parameter is of the varchar type.

The varchar type.

Concatenate the values of the region and request_method fields into one string.

- Sample field

```
region:cn-shanghai
time_local:14/Jul/2021:02:19:40
```

- Query statement

```
* | SELECT concat(region,'-',time_local)
```

- Query and analysis results

from_utf8 function

The from_utf8 function decodes a binary string into a UTF-8 encoded string.

- Invalid UTF-8 sequences are replaced with the default replacement character U+FFFD.

```
from_utf8()
```

- Invalid UTF-8 sequences are replaced with a custom string.

```
from_utf8(,replace_string)
```

Parameter	Description
	The value of this parameter is of the binary type.
replace_string	The value of this parameter is the custom string that you want to use. You can specify a single character or a space.

The varchar type.

- Decode the binary string 0x80 into a UTF-8 encoded string and replace invalid UTF-8 sequences in the result with the default replacement character U+FFFD. U+FFFD is displayed as .
  - Query statement

    ```
    * | SELECT from_utf8(from_base64('0x80'))
    ```

  - Query and analysis results

- Decode the binary string 0x80 into a UTF-8 encoded string and replace invalid UTF-8 sequences in the result with 0.
  - Query statement

    ```
    * | SELECT from_utf8(from_base64('0x80'),'0')
    ```

  - Query and analysis results

length function

The length function returns the length of a string.

```
length()
```

Parameter	Description
	The value of this parameter is of the varchar type.

The bigint type.

Calculate the length of the value of the http_user_agent field.

- Sample field

  ```
  http_user_agent:Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.2 (KHTML, like Gecko) Chrome/22.0.121
  6.0 Safari/537.2
  ```

- Query statement

  ```
  * | SELECT length(http_user_agent)
  ```

- Query and analysis results

levenshtein_distance function

The levenshtein_distance function returns the minimum edit distance between two strings.

```
levenshtein_distance(,)
```

Parameter	Description
	The value of this parameter is of the varchar type.
	The value of this parameter is of the varchar type.

The bigint type.

Query the minimum edit distance between the value of the instance_id field and the value of the owner_id field.

- Sample field

```
instance_id:i-01
owner_id:owner-01
```

- Query statement

```
* | SELECT levenshtein_distance(owner_id,instance_id)
```

- Query and analysis results

lower function

The lower function converts the characters in a string to lowercase letters.

```
lower()
```

Parameter	Description
	The value of this parameter is of the varchar type.

The varchar type.

Convert the characters in the value of the request_method field to lowercase letters.

- Sample field

```
request_method:GET
```

- Query statement

```
* | SELECT lower(request_method)
```

- Query and analysis results

lpad function

The lpad function left pads a string to a specified length by using a specified character and returns the result string.

```
lpad(,length,lpad_string)
```

Parameter	Description
	The value of this parameter is of the varchar type.
length	The value of this parameter is an integer that specifies the length of the result string. • If the length of a string is less than the value of the *length* parameter, the string is padded by using the specified character from the start of the string. • If the length of a string is greater than the value of the *length* parameter, only the n characters in the string are returned. n is specified by *length*.
lpad_string	The value of this parameter is the character that you want to use to pad a string.

The varchar type.

Pad the value of the instance_id field to 10 characters. If the value length is less than 10 characters, use 0 to pad the value from the start of the value.

- Sample field

  ```
  instance_id:i-01
  ```

- Query statement

  ```
  * | SELECT lpad(instance_id,10,'0')
  ```

- Query and analysis results

ltrim function

The ltrim function removes spaces from the start of a string.

```
ltrim()
```

Parameter	Description
	The value of this parameter is of the varchar type.

The varchar type.

Remove spaces from the start of the value of the region field.

- Sample field

```
region: cn-shanghai
```

- Query statement

```
* | SELECT ltrim(region)
```

- Query and analysis results

normalize function

The normalize function transforms a string by using the NFC normalization form.

```
normalize()
```

Parameter	Description
	The value of this parameter is of the varchar type.

The varchar type.

Transform the schön string by using the NFC normalization form.

- Query statement

```
* | SELECT normalize('schön')
```

- Query and analysis results

position function

The position function returns the position of a specified substring in a string.

```
position(sub_string in )
```

Parameter	Description
sub_string	The value of this parameter is the substring whose position you want to query.
	The value of this parameter is of the varchar type.

The integer type. Valid values start from 1.

Query the position of the cn substring in the value of the region field.

- Sample field

```
region:cn-shanghai
```

- Query statement

```
* | SELECT position('cn' in region)
```

- Query and analysis results

replace function

The replace function removes the matched characters from a string or replaces the matched characters in a string with specified characters.

- The following function removes the matched characters from a string.

```
replace(,sub_string)
```

- The following function replaces the matched characters in a string with specified characters.

```
replace(,sub_string,replace_string)
```

Parameter	Description
	The value of this parameter is of the varchar type.
sub_string	The value of this parameter is the substring that you want to match.
replace_string	The value of this parameter is the substring that you want to use to replace the matched substring.

The varchar type.

- Example 1: Replace cn in the value of the region field with China.
  - Sample field

    ```
    region:cn-shanghai
    ```

  - Query statement

    ```
    * | select replace(region,'cn','China')
    ```

  - Query and analysis results

- Example 2: Remove cn- from the value of the region field.
  - Sample field

    ```
    region:cn-shanghai
    ```

  - Query statement

    ```
    * | select replace(region,'cn-')
    ```

- Query and analysis results

reverse function

The reverse function returns a string in reverse order.

```
reverse()
```

Parameter	Description
	The value of this parameter is of the varchar type.

The varchar type.

Reverse the characters in the value of the request_method field.

- Sample field

```
request_method:GET
```

- Query statement

```
* | SELECT reverse(request_method)
```

- Query and analysis results

rpad function

The rpad function right pads a string to a specified length by using a specified character and returns the result string.

```
rpad(,length,rpad_string)
```

Parameter	Description
	The value of this parameter is of the varchar type.
length	The value of this parameter is an integer that specifies the length of the result string. • If the length of a string is less than the value of the *length* parameter, the string is padded by using the specified character from the end of the string. • If the length of a string is greater than the value of the *length* parameter, only the n characters in the string are returned. n is specified by *length*.
lpad_string	The value of this parameter is the character that you want to use to pad a string.

The varchar type.

Pad the value of the instance_id field to 10 characters. If the value length is less than 10 characters, use 0 to pad the value from the end of the value.

- Sample field

```
instance_id:i-01
```

- Query statement

```
* | SELECT rpad(instance_id,10,'0')
```

- Query and analysis results

rtrim function

The rtrim function removes spaces from the end of a string.

```
rtrim()
```

Parameter	Description
	The value of this parameter is of the varchar type.

The varchar type.

Remove spaces from the end of the value of the instance_id field.

- Sample field

```
instance_id:i-01
```

- Query statement

```
* | SELECT rtrim(instance_id)
```

- Query and analysis results

split function

The split function splits a string by using a specified delimiter and returns a set of substrings.

- The following function splits a string by using a specified delimiter and returns a set of substrings.

```
split(,delimeter)
```

- The following function splits a string by using a specified delimiter and returns a set of substrings. The number of substrings that can be generated is specified by *limit*.

```
split(,delimeter,limit)
```

Parameter	Description

Parameter	Description
	The value of this parameter is of the varchar type.
delimeter	The value of this parameter is the delimiter that you want to use.
limit	The value of this parameter is a positive integer. The value specifies the number of substrings that can be generated.

The array type.

- Example 1: Use a forward slash (/) to split the value of the request_uri field into four substrings and obtain a set of the substrings.
  - Sample field

    ```
    request_uri:/request/path-1/file-9
    ```

  - Query statement

    ```
    * | SELECT split(request_uri,'/')
    ```

  - Query and analysis results

- Example 2: Use a forward slash (/) to split the value of the request_uri field into three substrings and obtain a set of the substrings.
  - Sample field

    ```
    request_uri:/request/path-1/file-9
    ```

  - Query statement

    ```
    * | SELECT split(request_uri,'/',3)
    ```

  - Query and analysis results

split_part function

The split_part function splits a string by using a specified delimiter and returns the substring at a specified position.

```
split_part(,delimeter,part)
```

Parameter	Description
	The value of this parameter is of the varchar type.
delimeter	The value of this parameter is the delimiter that you want to use.
part	The value of this parameter is a positive integer.

The varchar type.

Use a question mark (?) to split the value of the request_uri field and obtain the first substring, which is a file path. Then, measure the number of requests that correspond to each path.

- Query statement

```
* | SELECT count(*) AS PV, split_part(request_uri, '?', 1) AS Path GROUP BY Path ORDER BY pv DESC
LIMIT 3
```

- Query and analysis results

PV	Path
4355	/request/path-1/file-5
4328	/request/path-1/file-1
4296	/request/path-2/file-3

split_to_map function

The split_to_map function splits a string by using the first specified delimiter, and then splits the string by using the second specified delimiter.

```
split_to_map(,delimiter01,delimiter02)
```

Parameter	Description
	The value of this parameter is of the varchar type.
delimeter01	The value of this parameter is the delimiter that you want to use.
delimeter02	The value of this parameter is the delimiter that you want to use.

The map type.

Use a comma (,) and a colon (:) to split the value of the time field to obtain a value of the map type.

- Sample field

```
time:upstream_response_time:"80", request_time:"40"
```

- Query statement

```
* | SELECT split_to_map(time,',',':')
```

- Query and analysis results

_col0
{"request_time":"\"40\"","upstream_response_time":"\"80\""}
{"request_time":"\"40\"","upstream_response_time":"\"80\""}

strpos function

The strpos function returns the position of a specified substring in a string. This function is equivalent to the position function.

```
strpos(,sub_string)
```

Parameter	Description
	The value of this parameter is of the varchar type.
sub_string	The value of this parameter is the substring whose position you want to query.

The integer type. Valid values start from 1.

Query the position of the letter H in the value of the server_protocol field.

- Query statement

```
* | SELECT strpos(server_protocol,'H')
```

- Query and analysis results

_col0
1
1

substr function

The substr function returns the substring at a specified position in a string.

- The following function returns the substring at a specified position in a string.

```
substr(,start)
```

- The following function returns the substring at a specified position in a string. The length of the substring is specified.

```
substr(,start,length)
```

Parameter	Description
	The value of this parameter is of the varchar type.
start	The value of this parameter is the start position from which you want to extract a substring. Valid values start from 1.
length	The value of this parameter is the length of the substring.

The varchar type.

Extract the first four characters (HTTP) from the value of the server_protocol field. Then, measure the number of requests that use the HTTP protocol.

- Sample field

```
server_protocol:HTTP/2.0
```

- Query statement

```
* | SELECT substr(server_protocol,1,4) AS protocol, count(*) AS count GROUP BY server_protocol
```

- Query and analysis results

protocol	count
HTTP	9078

to_utf8 function

The to_utf8 function converts a string to a UTF-8 representation.

```
to_utf8()
```

Parameter	Description
	The value of this parameter is of the varchar type.

The varbinary type.

Convert the log string to a UTF-8 representation.

- Query statement

```
* | SELECT to_utf8('log')
```

- Query and analysis results

trim function

The trim function removes spaces from the start and end of a string.

```
trim()
```

Parameter	Description
	The value of this parameter is of the varchar type.

The varchar type.

Remove spaces from the start and end of the value of the instance_id field.

- Sample field

```
instance_id: i-01
```

- Query statement

```
* | SELECT trim(instance_id)
```

- Query and analysis results

upper function

The upper function converts the characters in a string to uppercase letters.

```
upper()
```

Parameter	Description
	The value of this parameter is of the varchar type.

The varchar type.

Convert the characters in the value of the region field to uppercase letters.

- Sample field

```
region:cn-shanghai
```

- Query statement

```
* | SELECT upper(region)
```

- Query and analysis results

8.1.4. Date and time functions

Log Service allows you to use different types of date and time functions to process log data. The functions include time functions, date functions, date and time extraction functions, time interval functions, and time series padding functions. You can use the functions to convert the format of date and time in logs. You can also use the functions to group and aggregate logs. This topic describes the syntax of date and time functions. This topic also provides examples on how to use date and time functions.

Notice

- The timestamp of a log in Log Service is accurate to the second. Therefore, you can specify the time format only to the second. For more information, see Formats.
- You need to specify the time format only for the time in a time string. You do not need to specify the time format for other parameters such as the time zone. For more information, see Formats.
- Each log in Log Service contains the reserved __time__ field. The value of the field is a UNIX timestamp. Example: 1592374067, which indicates 2020-06-17 14:07:47.
- If you want to use strings in analytic statements, you must enclose the strings in single quotation marks (''). Strings that are not enclosed or are enclosed in double quotation marks ("") indicate field names or column names. For example, 'status' indicates the status string, and status or "status" indicates the status log field.

Type	Function	Syntax	Description
	current_date function	current_date	Returns the current date.
	current_time function	current_time	Returns the current time and time zone.
	current_timestamp function	current_timestamp	Returns the current date, time, and time zone.
	current_timezone function	current_timezone()	Returns the current time zone.

Type	Function	Syntax	Description
Date and time functions	date function	date()	Returns the date part of a datetime expression.
	date_format function	date_format(,*format*)	Converts a datetime expression that can return a timestamp value to a datetime expression in a specified format.
	date_parse function	date_parse(,*format*)	Converts a datetime string to a datetime expression that can return a timestamp value and is in a specified format.
	from_iso8601_date function	from_iso8601_date()	Converts a date expression in the ISO 8601 format to a date expression that can return a date value.
	from_iso8601_timestamp function	from_iso8601_timestamp()	Converts a datetime expression in the ISO 8601 format to a datetime expression that can return a timestamp value.
	from_unixtime function	from_unixtime()	Converts a UNIX timestamp to a datetime expression that can return a timestamp value and does not contain a time zone.
		from_unixtime(,*time zone*)	Converts a UNIX timestamp to a datetime expression that can return a timestamp value and contains a time zone.
		from_unixtime(,*hours*,*minutes*)	Converts a UNIX timestamp to a datetime expression that can return a timestamp value and contains a time zone. In the datetime expression, the values for *hours* and *minutes* indicate the offset of the time zone.
	localtime function	localtime	Returns the local time.
	localtimestamp function	localtimestamp	Returns the local date and time.
	now function	now()	Returns the current date and time. This function is equivalent to the current_timestamp function.
	to_iso8601 function	to_iso8601()	Converts a datetime expression that can return a date or timestamp value to a datetime expression in the ISO 8601 format.

Type	Function	Syntax	Description
	to_unixtime function	to_unixtime()	Converts a datetime expression that can return a timestamp value to a UNIX timestamp.
Date and time extraction functions	day function	day()	Returns the day of the month from a datetime expression. This function is equivalent to the day_of_month function.
	day_of_month function	day_of_month()	Returns the day of the month from a datetime expression. This function is equivalent to the day function.
	day_of_week function	day_of_week()	Returns the day of the week from a datetime expression. This function is equivalent to the dow function.
	day_of_year function	day_of_year()	Returns the day of the year from a datetime expression. This function is equivalent to the doy function.
	dow function	dow()	Returns the day of the week from a datetime expression. This function is equivalent to the day_of_week function.
	doy function	doy()	Returns the day of the year from a datetime expression. This function is equivalent to the day_of_year function.
	extract function	extract(*field* from)	Returns the specified *field* from a datetime expression. The field can be a date or time.
	hour function	hour()	Returns the hour of the day from a datetime expression. The 24-hour clock is used.
	minute function	minute()	Returns the minute of the hour from a datetime expression.
	month function	month()	Returns the month of the year from a datetime expression.
	quarter function	quarter()	Returns the quarter of the year on which a specified date falls.

Type	Function	Syntax	Description
	second function	second()	Returns the second of the minute from a datetime expression.
	timezone_hour function	timezone_hour()	Returns the offset of the time zone in hours.
	timezone_minute function	timezone_minute()	Returns the offset of the time zone in minutes.
	week function	week()	Returns the week of the year on which a specified date falls. This function is equivalent to the week_of_year function.
	week_of_year function	week_of_year()	Returns the week of the year on which a specified date falls. This function is equivalent to the week function.
	year function	year()	Returns the year of a specified date.
	year_of_week function	year_of_week()	Returns the year on which a specified date falls in the ISO week date system. This function is equivalent to the yow function.
	yow function	yow()	Returns the year on which a specified date falls in the ISO week date system. This function is equivalent to the year_of_week function.
Time interval functions	date_trunc function	date_trunc(unit,)	Truncates a datetime expression based on the time unit that you specify. The expression can be truncated based on the millisecond, second, minute, hour, day, month, or year.
	date_add function	date_add(*unit*, *N*,)	Adds N to the value of the field based on the unit that you specify.
	date_diff function	date_diff(*unit*, ,)	Returns the difference between two time expressions. For example, you can calculate the difference between and based on the time unit that you specify.

Type	Function	Syntax	Description
Time series padding function	time_series function	time_series(, *window*, *format*, *padding_data*)	Adds a value to the field that has no value returned in the specified time window.

current_date function

The current_date function returns the current date. The return value is in the YYYY-MM-DD format.

```
current_date
```

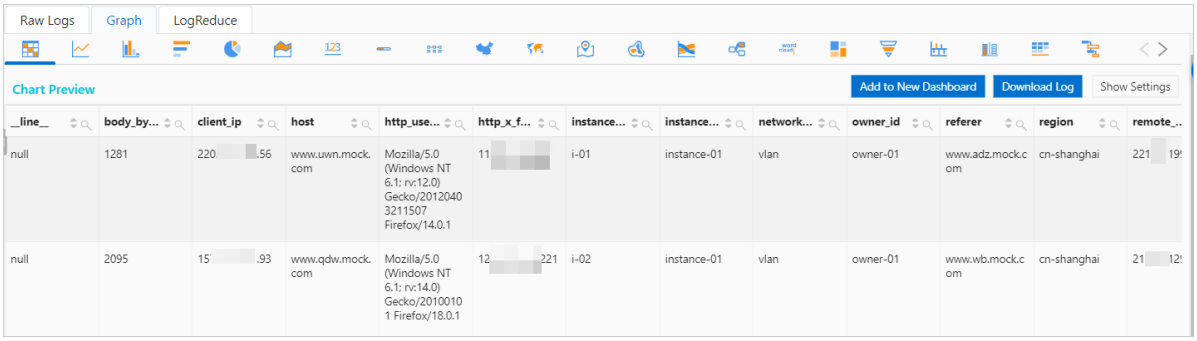
The date type.

Query the logs of the previous day.

- Query statement

```
* |
SELECT
  *
FROM  log
WHERE
  __time__ < to_unixtime(current_date)
  AND __time__ > to_unixtime(date_add('day', -1, current_date))
```

- Query and analysis results



current_time function

The current_time function returns the current time and time zone. The return value is in the HH:MM:SS.Ms Time_zone format.

```
current_time
```

The time type.

Query the current time and time zone.

- Query statement

```
* | select current_time
```

- Query and analysis results

current_timestamp function

The current_timestamp function returns the current date, time, and time zone. The return value is in the YYYY-MM-DD HH:MM:SS.Ms Time_zone format.

```
current_timestamp
```

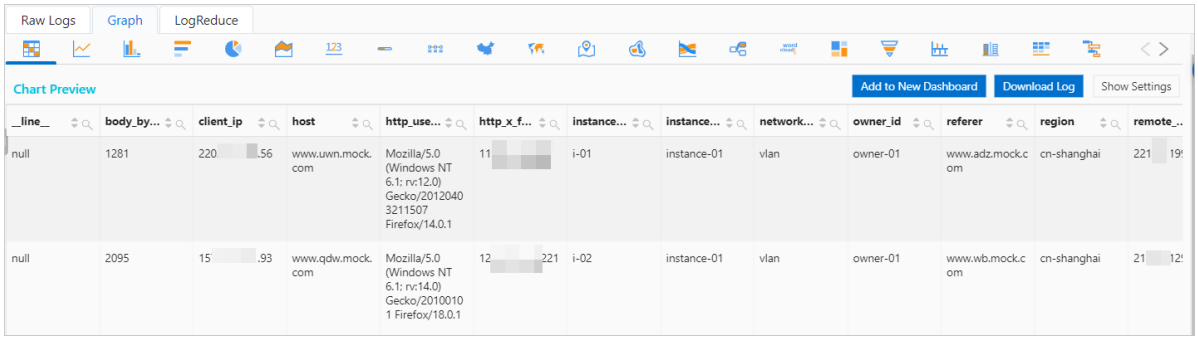
The timestamp type.

Query the logs of the previous day.

- Query statement

```
* |
SELECT
  *
FROM  log
WHERE
  __time__ < to_unixtime(current_timestamp)
  AND __time__ > to_unixtime(date_add('day', -1, current_timestamp))
```

- Query and analysis results



current_timezone function

The current_timezone function returns the current time zone.

```
current_timezone()
```

The varchar type.

Query the current time zone.

- Query statement

```
* | select current_timezone()
```

- Query and analysis results

date function

The date function returns the date part of a datetime expression. This function is equivalent to the `cast( as date)` function. For more information, see Data type conversion functions.

```
date()
```

Parameter	Description
	The value of this parameter is of the date or timestamp type.

The date type.

Use the current_timestamp function to obtain the current date and time. Then, use the date function to obtain the date part of the current date and time.

- Query statement

```
* | SELECT current_timestamp, date(current_timestamp)
```

- Query and analysis results

date_format function

The date_format function converts a datetime expression that can return a timestamp value to a datetime expression in a specified format.

```
date_format(,format)
```

Parameter	Description
	The value of this parameter is a datetime expression that can return a timestamp value.
format	The format of the datetime expression to which you want to convert a datetime expression that can return a timestamp value. For more information, see Formats.

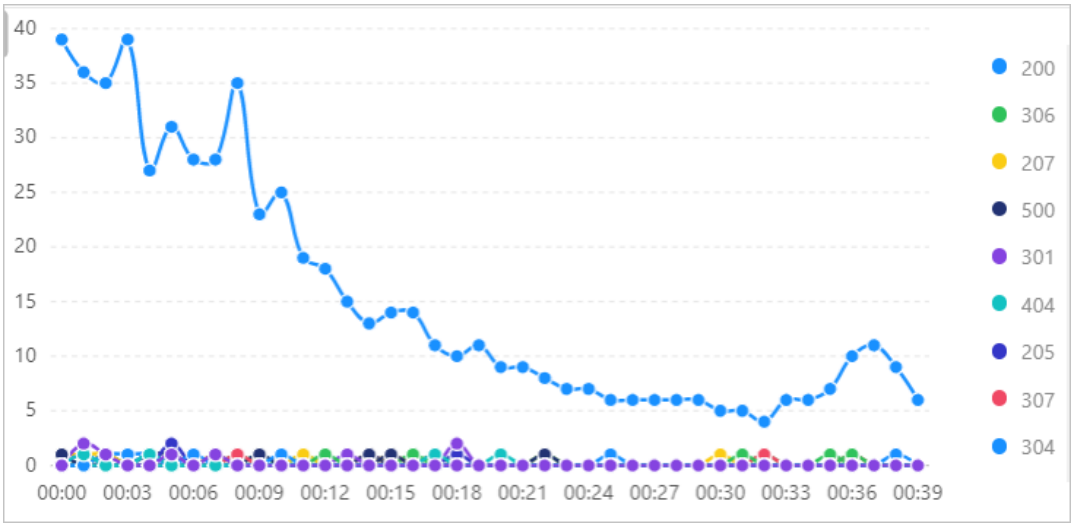
The varchar type.

Query the status of NGINX requests, calculate the number of NGINX requests, and then display the query and analysis results in chronological order. To do this, use the date_trunc function to truncate the log time by minute and use the date_format function to convert the time to the `%H:%i` format. Then, calculate the number of requests for each status code per minute and display the query and analysis results in a flow chart.

- Query statement

```
* |
SELECT
  date_format(date_trunc('minute', __time__), '%H:%i') AS time,
  COUNT(1) AS count,
  status
GROUP BY
  time,
  status
ORDER BY
  time
```

- Query and analysis results



date_parse function

The date_parse function converts a datetime string to a datetime expression that can return a timestamp value and is in a specified format.

```
date_parse(,format)
```

Parameter	Description
	The value of this parameter is a datetime string.
format	The format of the datetime expression that can return a timestamp value to which you want to convert the datetime string. For more information, see Formats.

The timestamp type.

Convert the values of the StartTime and EndTime fields to datetime expressions that can return a timestamp value and calculate the difference between the two datetime expressions.

- Query statement

```
*|
SELECT
    date_parse(StartTime, '%Y-%m-%d %H:%i') AS "StartTime",
    date_parse(EndTime, '%Y-%m-%d %H:%i') AS "EndTime",
    date_diff('hour', StartTime, EndTime) AS "Time difference (hour)"
```

- Query and analysis results

StartTime	EndTime	Time Difference (Hours)
2021-07-21 20:00:00.000	2021-07-21 21:00:00.000	1

from_iso8601_date function

The from_iso8601_date function converts a date expression in the ISO 8601 format to a date expression that can return a date value. The return value is in the YYYY-MM-DD format.

```
from_iso8601_date()
```

Parameter	Description
	The value of this parameter is a date expression in the ISO 8601 format.

The date type.

Convert the value of the time field to a date expression that can return a date value

- Sample field

```
time:2020-05-03
```

- Query statement

```
* | select from_iso8601_date(time)
```

- Query and analysis results

_col0
2020-05-03
2020-05-03
2020-05-03

from_iso8601_timestamp function

The from_iso8601_timestamp function converts a datetime expression in the ISO 8601 format to a datetime expression that can return a timestamp value. The return value is in the YYYY-MM-DD HH:MM:SS.Ms Time_zone format.

```
from_iso8601_timestamp()
```

Parameter	Description
	The value of this parameter is a datetime expression in the ISO 8601 format.

The timestamp type.

Convert the value of the time field to a datetime expression that can return a timestamp value.

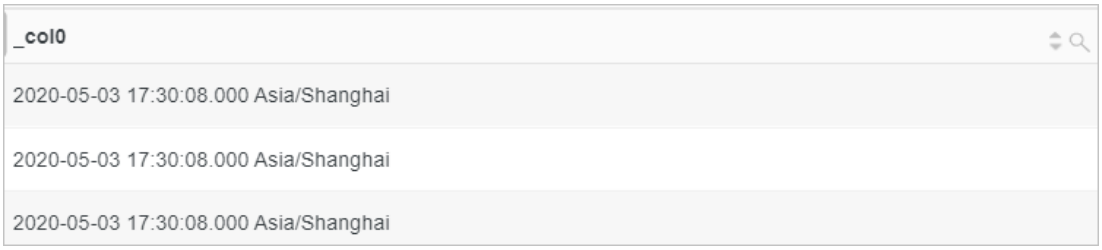
- Sample field

```
time:2020-05-03T17:30:08
```

- Query statement

```
* | select from_iso8601_timestamp(time)
```

- Query and analysis results



_col0
2020-05-03 17:30:08.000 Asia/Shanghai
2020-05-03 17:30:08.000 Asia/Shanghai
2020-05-03 17:30:08.000 Asia/Shanghai

from_unixtime function

The from_unixtime function converts a UNIX timestamp to a datetime expression that can return a timestamp value. The return value is in the YYYY-MM-DD HH:MM:SS.Ms or YYYY-MM-DD HH:MM:SS.Ms Time_zone format.

- The following function converts a UNIX timestamp to a datetime expression that can return a timestamp value and does not contain a timezone.

```
from_unixtime()
```

- The following function converts a UNIX timestamp to a datetime expression that can return a timestamp value and contains a timezone.

```
from_unixtime(,time zone)
```

- The following function converts a UNIX timestamp to a datetime expression that can return a timestamp value and contains a timezone. In the datetime expression, the values for *hours* and *minutes* indicate the offset of the time zone.

```
from_unixtime(,hours,minutes)
```

Parameter	Description
	The value of this parameter is a UNIX timestamp.
time zone	The time zone. Example: Asia/shanghai.
hours	The offset of the time zone in hours. Examples: +07 and -09.
minutes	The offset of the time zone in minutes. Examples: +30 and -45.

The timestamp type.

Convert the value of the time field to a datetime expression that can return a timestamp value and contains a timezone.

- Sample field

```
time:1626774758
```

- Query statement

```
* | select from_unixtime(time,'Asia/shanghai')
```

- Query and analysis results

localtime function

The localtime function returns the local time. The return value is in the HH:MM:SS.Ms format.

```
localtime
```

The time type.

Query the local time.

- Query statement

```
* | select localtime
```

- Query and analysis results

localtimestamp function

The localtimestamp function returns the local date and time. The return value is in the YYYY-MM-DD HH:MM:SS.Ms Time_zone format.

```
localtimestamp
```

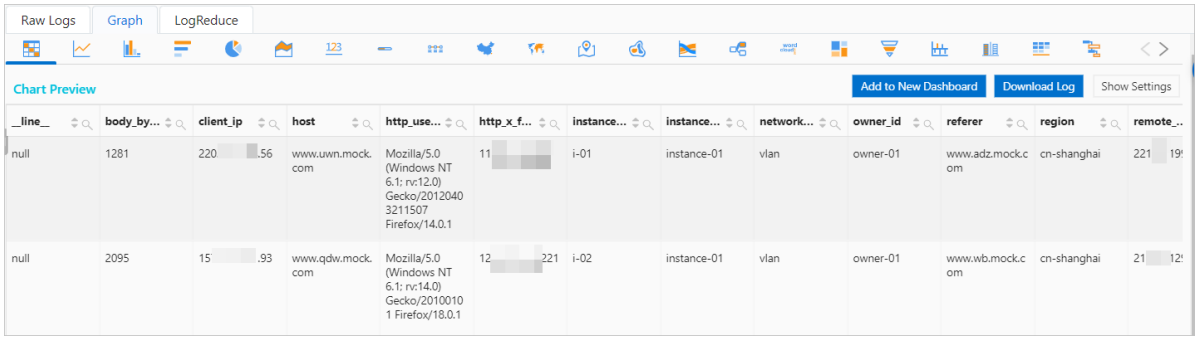
The timestamp type.

Query the logs of the previous day.

- Query statement

```
* |
SELECT
  *
FROM  log
WHERE
  __time__ < to_unixtime(localtimestamp)
  AND __time__ > to_unixtime(date_add('day', -1, localtimestamp))
```

- Query and analysis results



now function

The now function returns the current date and time. The return value is in the YYYY-MM-DD HH:MM:SS.Ms Time_zone format. This function is equivalent to the current_timestamp function.

```
now()
```

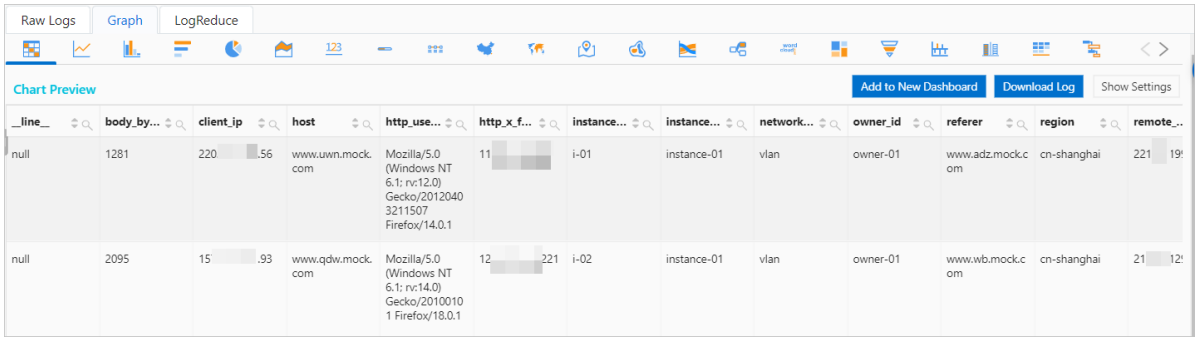
The timestamp type.

Query the logs of the previous day.

- Query statement

```
* |
SELECT
  *
FROM  log
WHERE
  __time__ < to_unixtime(now())
  AND __time__ > to_unixtime(date_add('day', -1, now()))
```

- Query and analysis results



to_iso8601 function

The to_iso8601 function converts a datetime expression that can return a date or timestamp value to a datetime expression in the ISO 8601 format.

```
to_iso8601()
```

Parameter	Description
	The value of this parameter is of the date or timestamp type.

The varchar type.

Use the current_timestamp function to obtain the current date and time. Then, use the to_iso8601 function to convert the current date and time to the ISO 8601 format.

- Query statement

```
* | select to_iso8601(current_timestamp) AS ISO8601
```

- Query and analysis results

to_unixtime function

The to_unixtime function converts a datetime expression that can return a timestamp value to a UNIX timestamp.

```
to_unixtime()
```

Parameter	Description
	The value of this parameter is a datetime expression that can return a timestamp value.

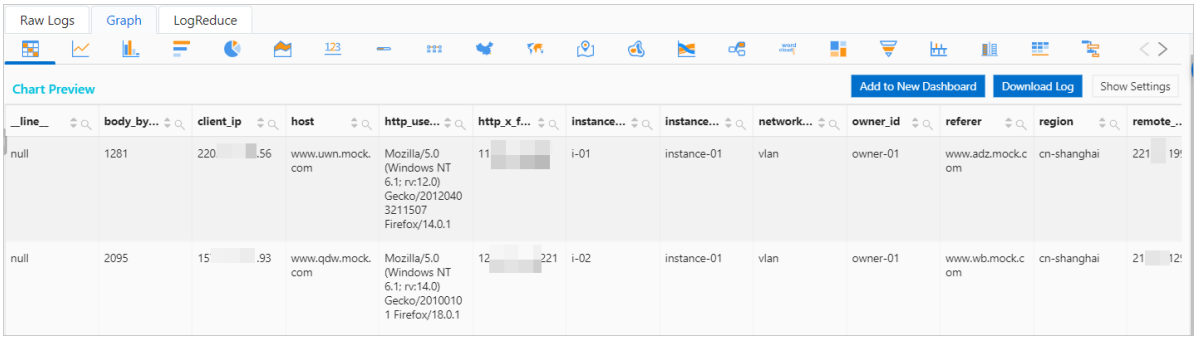
The double type.

Query the logs of the previous day.

- Query statement

```
* |
SELECT
  *
FROM  log
WHERE
  __time__ < to_unixtime(now())
  AND __time__ > to_unixtime(date_add('day', -1, now()))
```

- Query and analysis results



day function

The day function returns the day of the month from a datetime expression. This function is equivalent to the day_of_month function.

```
day()
```

Parameter	Description
	The value of this parameter is of the timestamp or date type.

The bigint type.

Use the current_date function to obtain the current date. Then, use the day function to obtain the day of the month based on the current date.

- Query statement

```
* | SELECT current_date, day(current_date)
```

- Query and analysis results

_col0	_col1
2021-09-07	7

day_of_month function

The day_of_month function returns the day of the month from a datetime expression. This function is equivalent to the day function.

```
day_of_month()
```

Parameter	Description
	The value of this parameter is of the timestamp or date type.

The bigint type.

Use the current_date function to obtain the current date. Then, use the day_of_month function to obtain the day of the month based on the current date.

- Query statement

```
* | SELECT current_date, day_of_month(current_date)
```

- Query and analysis results

_col0	_col1
2021-09-07	7

day_of_week function

The day_of_week function returns the day of the week from a datetime expression.

```
day_of_week()
```

Parameter	Description
	The value of this parameter is of the timestamp or date type.

The bigint type.

Use the current_date function to obtain the current date. Then, use the day_of_week function to obtain the day of the week based on the current date.

- Query statement

```
* | SELECT current_date, day_of_week(current_date)
```

- Query and analysis results

_col0	_col1
2021-09-07	2

day_of_year function

The day_of_year function returns the day of the year from a datetime expression.

```
day_of_year()
```

Parameter	Description
	The value of this parameter is of the timestamp or date type.

The bigint type.

Use the current_date function to obtain the current date. Then, use the day_of_year function to obtain the day of the year based on the current date.

- Query statement

```
* | SELECT current_date, day_of_year(current_date)
```

- Query and analysis results

_col0	_col1
2021-09-07	250

dow function

The dow function returns the day of the week from a datetime expression. This function is equivalent to the day_of_week function.

```
dow()
```

Parameter	Description
	The value of this parameter is of the timestamp or date type.

The bigint type.

Use the current_date function to obtain the current date. Then, use the dow function to obtain the day of the week based on the current date.

- Query statement

```
* | SELECT current_date, dow(current_date)
```

- Query and analysis results

_col0	_col1
2021-09-07	2

doy function

The doy function returns the day of the year from a datetime expression. This function is equivalent to the day_of_year function.

```
doy()
```

Parameter	Description
	The value of this parameter is of the timestamp or date type.

The bigint type.

Use the current_date function to obtain the current date. Then, use the doy function to obtain the day of the year based on the current date.

- Query statement

```
* | SELECT current_date, doy(current_date)
```

- Query and analysis results

_col0	_col1
2021-09-07	250

extract function

The extract function returns the specified *field* from a datetime expression. The field can be a date or a time.

```
extract(field from )
```

Parameter	Description
field	Valid values: year, quarter, month, week, day, day_of_month, day_of_week, dow, day_of_year, doy, year_of_week, yow, hour, minute, second, timezone_hour, and timezone_minute.
	The value of this parameter is of the date, time, timestamp, or interval (actual varchar(9)) type.

The bigint type.

Use the current_date function to obtain the current date. Then, use the extract function to obtain the year of the current date.

- Query statement

```
* | SELECT extract(year from current_date)
```

- Query and analysis results

_col0
2021

hour function

The hour function returns the hour of the day from a datetime expression. The 24-hour clock is used.

```
hour()
```

Parameter	Description
	The value of this parameter is of the timestamp type.

The bigint type.

Use the current_timestamp function to obtain the current date and time. Then, use the hour function to obtain the hour of the day based on the current time.

- Query statement

```
* | SELECT current_timestamp, hour(current_timestamp)
```

- Query and analysis results

_col0	_col1
2021-09-07 11:00:48.413 Asia/Shanghai	11

minute function

The minute function returns the minute of the hour from a datetime expression.

```
minute()
```

Parameter	Description
	The value of this parameter is of the timestamp type.

The bigint type.

Use the current_timestamp function to obtain the current date and time. Then, use the minute function to obtain the minute of the hour based on the current time.

- Query statement

```
* | SELECT current_timestamp, minute(current_timestamp)
```

- Query and analysis results

_col0	_col1
2021-09-07 11:17:11.676 Asia/Shanghai	17

month function

The month function returns the month of the year from a datetime expression.

```
month()
```

Parameter	Description
	The value of this parameter is of the date or timestamp type.

The bigint type.

Use the current_timestamp function to obtain the current date and time. Then, use the month function to obtain the month of the year based on the current date.

- Query statement

```
* | SELECT current_timestamp, month(current_timestamp)
```

- Query and analysis results

_col0	_col1
2021-09-07 11:13:39.230 Asia/Shanghai	9

quarter function

The quarter function returns the quarter of the year on which a specified date falls.

```
quarter()
```

Parameter	Description
	The value of this parameter is of the date or timestamp type.

The bigint type.

Use the current_timestamp function to obtain the current date and time. Then, use the quarter function to obtain the quarter of the year on which the current date falls.

- Query statement

```
* | SELECT current_timestamp,quarter(current_timestamp)
```

- Query and analysis results

_col0	_col1
2021-09-07 11:26:08.720 Asia/Shanghai	3

second function

The second function returns the second of the minute from a datetime expression.

```
second()
```

Parameter	Description
	The value of this parameter is of the timestamp type.

The bigint type.

Use the current_timestamp function to obtain the current date and time. Then, use the second function to obtain the second of the minute based on the current time.

- Query statement

```
* | SELECT current_timestamp,second(current_timestamp)
```

- Query and analysis results

_col0	_col1
2021-09-07 11:32:41.695 Asia/Shanghai	41

timezone_hour function

The timezone_hour function returns the offset of the time zone in hours.

```
timezone_hour()
```

Parameter	Description
	The value of this parameter is of the timestamp type.

The bigint type.

Use the current_timestamp function to obtain the current date and time. Then, use the timezone_hour function to obtain the offset of the time zone to which the current time belongs in hours.

- Query statement

```
* | SELECT current_timestamp,timezone_hour(current_timestamp)
```

- Query and analysis results

_col0	_col1
2021-09-07 11:35:51.605 Asia/Shanghai	8

timezone_minute function

The timezone_minute function returns the offset of the time zone in minutes.

```
timezone_minute()
```

Parameter	Description
	The value of this parameter is of the timestamp type.

The bigint type.

Use the current_timestamp function to obtain the current date and time. Then, use the timezone_minute function to obtain the offset of the time zone to which the current time belongs in minutes.

- Query statement

```
* | SELECT current_timestamp,timezone_minute(current_timestamp)
```

- Query and analysis results

_col0	_col1
2021-09-07 11:41:48.818 Asia/Shanghai	0

week function

The week function returns the week of the year on which a specified date falls. This function is equivalent to the week_of_year function.

```
week()
```

Parameter	Description
	The value of this parameter is of the date or timestamp type.

The bigint type.

Use the current_timestamp function to obtain the current date and time. Then, use the week function to obtain the week of the year on which the current date falls.

- Query statement

```
* | SELECT current_timestamp,week(current_timestamp)
```

- Query and analysis results

_col0	_col1
2021-09-07 13:40:04.940 Asia/Shanghai	36

week_of_year function

The week_of_year function returns the week of the year on which a specified date falls. This function is equivalent to the week function.

```
week_of_year()
```

Parameter	Description
	The value of this parameter is of the date or timestamp type.

The bigint type.

Use the current_timestamp function to obtain the current date and time. Then, use the week_of_year function to obtain the week of the year on which the current date falls.

- Query statement

```
* | SELECT current_timestamp,week_of_year(current_timestamp)
```

- Query and analysis results

_col0	_col1
2021-09-07 13:40:04.940 Asia/Shanghai	36

year function

The year function returns the year of a specified date.

```
year()
```

Parameter	Description
	The value of this parameter is of the date or timestamp type.

The bigint type.

Use the current_timestamp function to obtain the current date and time. Then, use the year function to obtain the year of the current date.

- Query statement

```
* | SELECT current_timestamp,year(current_timestamp)
```

- Query and analysis results

_col0	_col1
2021-09-07 13:49:47.845 Asia/Shanghai	2021

year_of_week function

The year_of_week function returns the year on which a specified date falls in the ISO week date system. This function is equivalent to the yow function.

```
year_of_week()
```

Parameter	Description
	The value of this parameter is of the date or timestamp type.

The bigint type.

Use the current_timestamp function to obtain the current date and time. Then, use the year_of_week function to obtain the year on which the current date falls in the ISO week date system.

- Query statement

```
* | SELECT current_timestamp,year_of_week(current_timestamp)
```

- Query and analysis results

_col0	_col1
2021-09-07 13:57:25.576 Asia/Shanghai	2021

yow function

The yow function returns the year on which a specified date falls in the ISO week date system. This function is equivalent to the year_of_week function.

```
yow()
```

Parameter	Description
	The value of this parameter is of the date or timestamp type.

The bigint type.

Use the current_timestamp function to obtain the current date and time. Then, use the yow function to obtain the year on which the current date falls in the ISO week date system.

- Query statement

```
* | SELECT current_timestamp,yow(current_timestamp)
```

- Query and analysis results

_col0	_col1
2021-09-07 13:57:25.576 Asia/Shanghai	2021

date_trunc function

The date_trunc function truncates a datetime expression based on the time unit that you specify. The expression can be truncated based on the millisecond, second, minute, hour, day, month, or year. This function is often used in scenarios that require statistical analysis by time.

```
date_trunc(unit,x)
```

Parameter	Description
unit	The unit of time. Valid values: millisecond, second, minute, hour, day, week, month, quarter, and year. For more information, see Units.
	The value of this parameter is a datetime expression.

Note The date_trunc function allows you to measure statistics only based on a fixed interval. The interval is determined by the time unit that you specify in the function. The time unit includes the minute or hour. If you want to measure statistics based on a custom interval, we recommend that you perform a mathematical modulo operation to group data. For example, a mathematical modulo operation is performed to group data at 5-minute intervals.

```
* | SELECT count(1) AS pv,  __time__ - __time__ %300 AS time GROUP BY time LIMIT 100
```

Same as the data type of the parameter value.

Calculate an average request duration by minute. Then, group and sort the durations in chronological order.

- Query statement

```
* |
SELECT
  date_trunc('minute', __time__) AS time,
  truncate (avg(request_time)) AS avg_time,
  current_date AS date
GROUP BY
  time
ORDER BY
  time DESC
LIMIT
  100
```

- Query and analysis results

time	avg_time	date
2021-07-21 11:26:00.000	47.0	2021-07-21
2021-07-21 11:25:00.000	44.0	2021-07-21
2021-07-21 11:24:00.000	44.0	2021-07-21

date_add function

The date_add function adds a specified interval to or subtract a specified interval from a date or time.

```
date_add(unit, n,)
```

Parameter	Description
unit	The unit of time. Valid values: millisecond, second, minute, hour, day, week, month, quarter, and year. For more information, see Units.
n	The time interval.
	The value of this parameter is a datetime expression that can return a timestamp value.

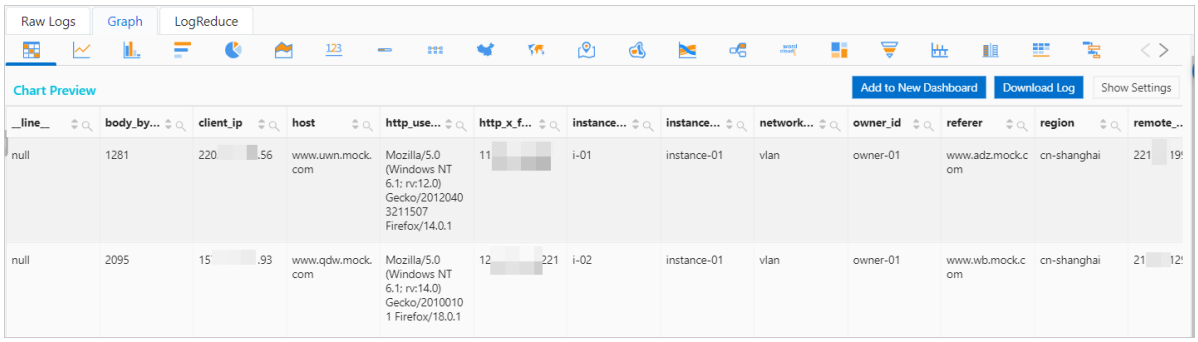
The timestamp type.

Query the logs of the previous day.

- Query statement

```
* |
SELECT
  *
FROM  log
WHERE
  __time__ < to_unixtime(current_timestamp)
  AND __time__ > to_unixtime(date_add('day', -1, current_timestamp))
```

- Query and analysis results



date_diff function

The date_diff function returns the difference between two dates or points in time.

```
date_diff(unit, , )
```

Parameter	Description
unit	The unit of time. Valid values: millisecond, second, minute, hour, day, week, month, quarter, and year. For more information, see Units.
	The value of this parameter is a datetime expression that can return a timestamp value.
	The value of this parameter is a datetime expression that can return a timestamp value.

The bigint type.

Calculate the runtime duration of a server based on the UsageStartTime and UsageEndTime fields.

- Query statement

```
* | SELECT  date_diff('hour', UsageStartTime, UsageEndTime) AS "Time difference (hour)"
```

- Query and analysis results

time_series function

The time_series function adds a value to the field that has no value returned in the specified time window.

Notice You must use the time_series function together with the GROUP BY and ORDER BY clauses. You cannot use the DESC keyword in the ORDER BY clause to sort data.

```
time_series(, window_time, format, padding_data)
```

Parameter	Description
	The time column. Example: __time__. The value of this parameter is of the long or timestamp type.
window_time	The duration of the time window. Unit: s, m, h, and d. s indicates second, m indicates minute, h indicates hour, and d indicates day. Examples: 2h, 5m, and 3d.
format	The time format in which you want the function to return the value. For more information, see Formats.
padding_data	The value that you want to add. Valid values: • 0: The value 0 is added. • null: The value null is added. • last: The value of the previous point in time is added. • next: The value of the next point in time is added. • avg: The average value of the previous point in time and the next point in time is added.

The varchar type.

Add the value 0 to the fields that have no value returned during two hours.

- Query statement

```
* | select time_series(__time__, '2h', '%Y-%m-%d %H:%i:%s', '0')  as time, count(*) as num from lo
g group by time order by time
```

- Query and analysis results

time	num
2021-07-20 00:00:00	11602
2021-07-20 02:00:00	63089
2021-07-20 04:00:00	36583
2021-07-20 06:00:00	11135
2021-07-20 08:00:00	62746
2021-07-20 10:00:00	18314

References

- Formats

Format	Description
%a	The abbreviation for the day of the week. Examples: Sun and Sat.
%b	The abbreviation for the month of the year. Examples: Jan and Dec.
%c	The month. The value is of the numeric type. Valid values: 1 to 12.
%D	The day of the month. Examples: 0th, 1st, 2nd, and 3rd.
%d	The day of the month. The value is in the decimal format. Valid values: 01 to 31.
%e	The day of the month. The value is in the decimal format. Valid values: 1 to 31.
%H	The hour. The 24-hour clock is used.
%h	The hour. The 12-hour clock is used.
%I	The hour. The 12-hour clock is used.
%i	The minute. The value is of the numeric type. Valid values: 00 to 59.
%j	The day of the year. Valid values: 001 to 366.
%k	The hour. Valid values: 0 to 23.
%l	The hour. Valid values: 1 to 12.
%M	The full month name. Examples: January and December.
%m	The month. The value is of the numeric type. Valid values: 01 to 12.
%p	The abbreviation that indicates the morning or afternoon of the day. Valid values: AM and PM.
%r	The time. The 12-hour clock is used. The time is in the `hh:mm:ss AM/PM` format.

Format	Description
%S	The second. Valid values: 00 to 59.
%s	The second. Valid values: 00 to 59.
%T	The time. The 24-hour clock is used. The time is in the `hh:mm:ss` format.
%V	The week of the year. Sunday is the first day of a week. Valid values: 01 to 53.
%v	The week of the year. Monday is the first day of a week. Valid values: 01 to 53.
%W	The full name of the day of the week. Examples: Sunday and Saturday.
%w	The day of the week. The value 0 indicates Sunday.
%Y	The four-digit year. Example: 2020.
%y	The two-digit year. Example: 20.
%%	The escape character of the percent sign (%).

- Units

Unit	Description
millisecond	Millisecond
second	Second
minute	Minute
hour	Hour
day	Day
week	Week
month	Month
quarter	Quarter
year	Year

8.1.5. JSON functions

This topic describes the syntax of JSON functions. This topic also provides examples on how to use the functions.

The following table describes the JSON functions that are supported by Log Service.

Notice

- If you want to use strings in analytic statements, you must enclose the strings in single quotation marks (''). Strings that are not enclosed or are enclosed in double quotation marks ("") indicate field names or column names. For example, 'status' indicates the status string, and status or "status" indicates the status log field.
- If the value of a log field is of the JSON type and needs to be expanded to multiple rows, we recommend that you use UNNEST clauses. For more information, see UNNEST clause.
- If a string fails to be parsed into JSON data, null is returned.
- JSON logs are truncated during the collection process. When you use a JSON function to query and analyze the JSON logs, the system returns an error and stops the query and analysis process. To troubleshoot this issue, you can specify a try function in a query statement to capture the error. Then, the system can continue the query and analysis process. Example: `* select message, try(json_parse(message))`. For more information, see TRY function.

Function	Syntax	Description
json_array_contains function	json_array_contains(, *value*)	Checks whether a JSON array contains a specified value.
json_array_get function	json_array_get(, *index*)	Obtains the element that corresponds to an index in a JSON array.
json_array_length function	json_array_length()	Calculates the number of elements in a JSON array.
json_extract function	json_extract(, json_path)	Extracts a set of JSON values from a JSON object or a JSON array.
json_extract_scalar function	json_extract_scalar(, *json_path*)	Extracts a set of scalar values from a JSON object or a JSON array. The scalar values can be of the string, integer, or Boolean type. This function is similar to the json_extract function.
json_format function	json_format()	Converts JSON data to a string.
json_parse function	json_parse()	Converts a string to JSON data.
json_size function	json_size(, *json_path*)	Calculates the number of elements in a JSON object or a JSON array.

json_array_contains function

The json_array_contains function is used to check whether a JSON array contains a specified value.

```
json_array_contains(, value)
```

Parameter	Description
	The value of this parameter is a JSON array.
value	The numeric value.

The Boolean type.

Check whether the [1, 2, 3] JSON string contains 2.

- Query statement

```
*  SELECT json_array_contains('[1, 2, 3]', 2)
```

- Query and analysis result

json_array_get function

The json_array_get function is used to obtain the element that corresponds to an index in a JSON array.

```
json_array_get(, index)
```

Parameter	Description
	The value of this parameter is a JSON array.
index	The JSON index. The value of this parameter starts from 0.

The varchar type.

Return the element that corresponds to the index 1 in the ["a", [3, 9], "c"] JSON array.

- Query statement

```
*  SELECT json_array_get('["a", [3, 9], "c"]', 1)
```

- Query and analysis result

json_array_length function

The json_array_length function is used to calculate the number of elements in a JSON array.

```
json_array_length()
```

Parameter	Description
	The value of this parameter is a JSON array.

The bigint type.

- Example 1: Calculate the number of JSON elements in the value of the Results field.
  - Sample field

    ```
    Results:[{"EndTime":1626314920},{"FireResult":2}]
    ```

  - Query statement

    ```
    *  SELECT json_array_length(Results)
    ```

- Query and analysis result

- Example 2: Calculate the number of JSON elements in the value of the time field.
  - Sample field

```
time:["time_local","request_time","upstream_response_time"]
```

  - Query statement

```
*  SELECT json_array_length(time)
```

  - Query and analysis result

json_extract function

The json_extract function is used to extract a set of JSON values from a JSON object or a JSON array.

> **Notice** If the JSON data is invalid when you use the json_extract function, an error message appears. We recommend that you use the json_extract_scalar function.

```
json_extract(, json_path)
```

Parameter	Description
	The value of this parameter is a JSON object or a JSON array.
json_path	The JSON path. Format: $.store.book[0].title.

The string type in the JSON format.

Extract the value of the EndTime field from the Results field.

- Sample field

```
Results:[{"EndTime":1626314920},{"FireResult":2}]
```

- Query statement

```
*  SELECT json_extract(Results, '$.0.EndTime')
```

- Query and analysis result

json_extract_scalar function

The json_extract_scalar function is used to extract a set of scalar values from a JSON object or a JSON array. The scalar values can be of the string, integer, or Boolean type.

```
json_extract_scalar(, json_path)
```

Parameter	Description
	The value of this parameter is a JSON object or a JSON array.
json_path	The JSON path. Format: $.store.book[0].title.

The varchar type.

Extract the value of the RawResultCount field from the Results field. Then, convert the value to the bigint type for summation.

- Sample field

```
Results:[{"EndTime":1626314920},{"RawResultCount":1}]
```

- Query statement

```
*  SELECT sum(cast(json_extract_scalar(Results,'$.0.RawResultCount') AS bigint) )
```

- Query and analysis result

_col0
288

json_format function

The json_format function is used to convert JSON data to a string.

```
json_format()
```

Parameter	Description
	The value of this parameter is of the JSON type.

The varchar type.

Convert the [1,2,3] JSON array to the [1, 2, 3] string.

- Query statement

```
*  SELECT json_format(json_parse('[1, 2, 3]'))
```

- Query and analysis result

_col0
[1,2,3]

json_parse function

The json_parse function is used only to convert a string to JSON data and check whether the string matches the JSON format. In most cases, the json_parse function is of little significance. If you want to extract values from JSON data, we recommend that you use the json_extract_scalar function.

```
json_parse()
```

Parameter	Description
	The value of this parameter is a string.

The JSON type.

Convert the [1,2,3] string to the [1, 2, 3] JSON array.

- Query statement

```
* SELECT json_parse('[1, 2, 3]')
```

- Query and analysis result

json_size function

The json_size function is used to calculate the number of elements in a JSON object or a JSON array.

```
json_size(, json_path)
```

Parameter	Description
	The value of this parameter is a JSON object or a JSON array.
json_path	The JSON path. Format: $.store.book[0].title.

The bigint type.

Calculate the number of elements in the status field.

- Sample field

```
Results:[{"EndTime":1626314920,"FireResult":2,"RawResults":[{"_col0":"1094"}]}]
```

- Query statement

```
* SELECT json_size(Results, '$.0.RawResults')
```

- Query and analysis result

8.1.6. Regular expression functions

This topic describes the syntax of regular expression functions. This topic also provides examples on how to use the functions.

The following table describes the regular expression functions that are supported by Log Service.

Notice

Function	Syntax	Description
regexp_extract_all function	regexp_extract_all(, *regular expression*)	Extracts the substrings that match a specified regular expression from a specified string and returns an array of all matched substrings.
	regexp_extract_all(, *regular expression*, *n*)	Extracts the substrings that match a specified regular expression from a specified string and returns an array of substrings that match the nth capturing group in the regular expression.
regexp_extract function	regexp_extract(, *regular expression*)	Extracts the first substring that matches a specified regular expression from a specified string and returns the substring.
	regexp_extract(, *regular expression*, *n*)	Extracts the substrings that match a specified regular expression from a specified string and returns the first substring that matches the nth capturing group in the regular expression.
regexp_like function	regexp_like(, *regular expression*)	Checks whether a specified string matches a specified regular expression.
regexp_replace function	regexp_replace(, *regular expression*)	Deletes the substrings that match a specified regular expression from a specified string and returns the substrings that are not deleted.
	regexp_replace(, *regular expression*, *replace string*)	Replaces the substrings that match a specified regular expression in a specified string and returns a new string.
regexp_split function	regexp_split(, *regular expression*)	Splits a specified string into multiple substrings by using a specified regular expression and returns an array of the substrings.

regexp_extract_all function

The regexp_extract_all function is used to extract the substrings that match a specified regular expression from a specified string.

- To extract the substrings that match a specified regular expression from a specified string and return an array of all matched substrings, use the following syntax:

```
regexp_extract_all(, regular expression)
```

- To extract the substrings that match a specified regular expression from a specified string and return an array of substrings that match the nth capturing group in the regular expression, use the following syntax:

```
regexp_extract_all(, regular expression, n)
```

Parameter	Description

Parameter	Description
	The value of this parameter is of the varchar type.
regular expression	The regular expression that contains capturing groups. For example, `(\d)(\d)(\d)` indicates three capturing groups.
n	The nth capturing group. n is an integer that is greater than or equal to 1.

The array type.

- Example 1: Extract all numbers from the value of the server_protocol field.
  - Sample field

    ```
    server_protocol:HTTP/2.0
    ```

  - Query statement

    ```
    *| SELECT regexp_extract_all(server_protocol, '\d+')
    ```

  - Query and analysis result

_col0
["2","0"]

- Example 2: Extract "Chrome" from the value of the http_user_agent field and calculate the number of requests that are initiated by the Chrome browser.
  - Sample field

    ```
    http_user_agent:Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/14.0.803.0 Safari/535.1
    ```

  - Query statement

    ```
    *| SELECT regexp_extract_all(http_user_agent, '(Chrome)',1) AS Chrome, count(*) AS count GROUP BY Chrome
    ```

  - Query and analysis result

Chrome	count
["Chrome"]	103440

regexp_extract function

The regexp_extract function is used to extract the substrings that match a specified regular expression from a specified string.

- To extract the first substring that matches a specified regular expression from a specified string and return the substring, use the following syntax:

  ```
  regexp_extract(, regular expression)
  ```

- To extract the substrings that match a specified regular expression from a specified string and return the first substring that matches the nth capturing group in the regular expression, use the following syntax:

  ```
  regexp_extract(, regular expression, n)
  ```

Parameter	Description
	The value of this parameter is of the varchar type.
regular expression	The regular expression that contains capturing groups. For example, `(\d)(\d)(\d)` indicates three capturing groups.
n	The nth capturing group. n is an integer that is greater than or equal to 1.

The varchar type.

- Example 1: Extract the first number from the value of the server_protocol field.
  - Sample field

    ```
    server_protocol:HTTP/2.0
    ```

  - Query statement

    ```
    *|SELECT regexp_extract(server_protocol, '\d+')
    ```

  - Query and analysis result

_col0
2

- Example 2: Extract the file information from the value of the request_uri field and calculate the number of visits for each file.
  - Sample field

    ```
    request_uri:/request/path-3/file-5
    ```

  - Query statement

    ```
    * | SELECT regexp_extract(request_uri, '.*\/(file.*)', 1) AS file, count(*) AS count GROUP BY file
    ```

  - Query and analysis result

file	count
file-5	17127

regexp_like function

The regexp_like function is used to check whether a specified string matches a specified regular expression.

```
regexp_like(, regular expression)
```

Parameter	Description
	The value of this parameter is of the varchar type.
regular expression	The regular expression.

The Boolean type.

Check whether the value of the server_protocol field contains digits.

- Sample field

```
server_protocol:HTTP/2.0
```

- Query statement

```
*| select regexp_like(server_protocol, '\d+')
```

- Query and analysis result

regexp_replace function

The regexp_replace function is used to delete or replace the substrings that match a specified regular expression from or in a specified string.

- To delete the substrings that match a specified regular expression from a specified string and return the substrings that are not deleted, use the following syntax:

```
regexp_replace(, regular expression)
```

- To replace the substrings that match a specified regular expression in a specified string and return a new string, use the following syntax:

```
regexp_replace(, regular expression, replace string)
```

Parameter	Description
	The value of this parameter is of the varchar type.
regular expression	The regular expression.
replace string	The substring that is used to replace the substring that matches the regular expression.

The varchar type.

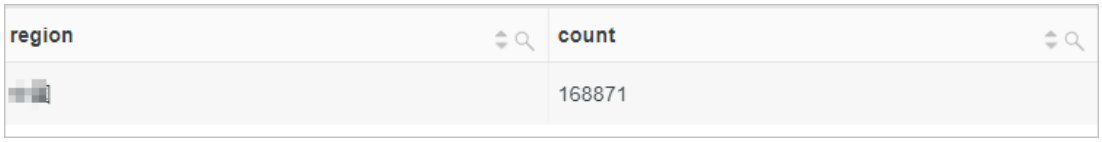
- Example 1: Replace the region name that starts with cn in the value of the region field with **China** and calculate the number of requests from China.
  - Sample field

```
region:cn-shanghai
```

  - Query statement

```
* | select regexp_replace(region, 'cn.*','China') AS region, count(*) AS count GROUP BY region
```

  - Query and analysis result



- Example 2: Delete the version number in the value of the server_protocol field and calculate the number of requests for each communication protocol.

- Sample field

```
server_protocol:HTTP/2.0
```

- Query statement

```
*| select regexp_replace(server_protocol, '.\d+') AS server_protocol, count(*) AS count GROUP BY server_protocol
```

- Query and analysis result

server_protocol	count
HTTP	168871

regexp_split function

The regexp_split function is used to split a specified string into multiple substrings by using a specified regular expression and return an array of the substrings.

```
regexp_split(, regular expression)
```

Parameter	Description
	The value of this parameter is of the varchar type.
regular expression	The regular expression.

The array type.

Split the value of the request_uri field with forward slashes (/).

- Sample field

```
request_uri:/request/path-0/file-7
```

- Query statement

```
* | SELECT regexp_split(request_uri,'/')
```

- Query and analysis result

_col0
["","request","path-0","file-7"]

8.1.7. Interval-valued comparison and periodicity-valued comparison functions

This topic describes the syntax of interval-valued comparison and periodicity-valued comparison functions. This topic also provides examples on how to use the functions.

The following table describes the interval-valued comparison and periodicity-valued comparison functions that are supported by Log Service.

Notice

Function	Syntax	Description
compare function	compare(,*n*)	Compares the calculation result of the current time period with the calculation result of a time period n seconds before.
	compare(,*n1*,*n2*,*n3*...)	Compares the calculation result of the current time period with the calculation results of time periods n1, n2, and n3 seconds before.
ts_compare function	ts_compare(,*n*)	Compares the calculation result of the current time period with the calculation result of a time period n seconds before. **Notice** The query and analysis results of the ts_compare function must be grouped by the time column by using the GROUP BY clause.
	ts_compare(,*n1*,*n2*,*n3*...)	Compares the calculation result of the current time period with the calculation results of time periods n1, n2, and n3 seconds before. **Notice** The query and analysis results of the ts_compare function must be grouped by the time column by using the GROUP BY clause.

compare function

The compare function is used to compare the calculation result of the current time period with the calculation result of a time period n seconds before.

- To compare the calculation result of the current time period with the calculation result of a time period n seconds before, use the following syntax:

```
compare(,n)
```

- To compare the calculation result of the current time period with the calculation results of time periods n1, n2, and n3 seconds before, use the following syntax:

```
compare(,n1,n2,n3...)
```

Parameter	Description
	The value of this parameter is of the double or long type.
n	The time window. Unit: seconds. Example: 3600 (1 hour), 86400 (1 day), 604800 (one week), or 31622400 (one year).

The returned result is a JSON array in the following format: [the current value, the value before n seconds, the ratio of the current value to the value of n seconds before, the UNIX timestamp before n seconds].

- Example 1: Calculate the ratio of the page views (PVs) of the current hour to the PVs of the same time period the day before.

Set the time range to **1 Hour(Time Frame)** and execute the following query statement. 86400 indicates the current time minus 86400 seconds (one day). log indicates the Logstore name.

- To display the query and analysis result in the form of a JSON array, execute the following query statement:
  - Query statement

```
* |
SELECT
  compare(PV, 86400)
FROM  (
    SELECT
      count(*) AS PV
    FROM      log
  )
```

  - Query and analysis result

_col0
[3337.0,3522.0,0.947473026689381]

    - **3337.0** indicates the PVs of the current 1 hour, for example, Dec 25, 2020, 14:00:00 ~ Dec 25, 2020, 15:00:00.
    - **3522.0** indicates the PVs of the same time period the day before, for example, Dec 24, 2020, 14:00:00 ~ Dec 24, 2020, 15:00:00.
    - **0.947473026689381** indicates the ratio of the PVs of the current hour to the PVs of the same time period the day before.
- To display the query and analysis result in multiple columns, execute the following query statement:
  - Query statement

```
* |
SELECT
  diff [1] AS today,
  diff [2] AS yesterday,
  diff [3] AS ratio
FROM  (
    SELECT
      compare(PV, 86400) AS diff
    FROM      (
        SELECT
          count(*) AS PV
        FROM          log
      )
  )
```

  - Query and analysis result

today	yesterday	ratio
3337.0	3522.0	0.947473026689381

    - **3337.0** indicates the PVs of the current 1 hour, for example, Dec 25, 2020, 14:00:00 ~ Dec 25, 2020, 15:00:00.
    - **3522.0** indicates the PVs of the same time period the day before, for example, Dec 24, 2020, 14:00:00 ~ Dec 24, 2020, 15:00:00.
    - **0.947473026689381** indicates the ratio of the PVs of the current hour to the PVs of the same time period the day before.

- Example 2: Calculate the ratio of the PVs of every hour today to the PVs of the same time period the day before and two days before.

Set the time range to **Today(Time Frame)** and execute the following query statement. 86400 indicates the current time minus 86400 seconds (one day). 172800 indicates the current time minus 172800 seconds (two days). log indicates the Logstore name. date_format(from_unixtime(__time__), '%H:00') indicates the returned time format.

- To display the query and analysis result in the form of a JSON array, execute the following query statement:
  - Query statement

```
* |
SELECT
  time,
  compare(PV, 86400, 172800) as diff
FROM  (
    SELECT
      count(*) as PV,
      date_format(from_unixtime(__time__), '%H:00') as time
    FROM      log
    GROUP BY
      time
  )
GROUP BY
  time
ORDER BY
  time
```

  - Query and analysis result

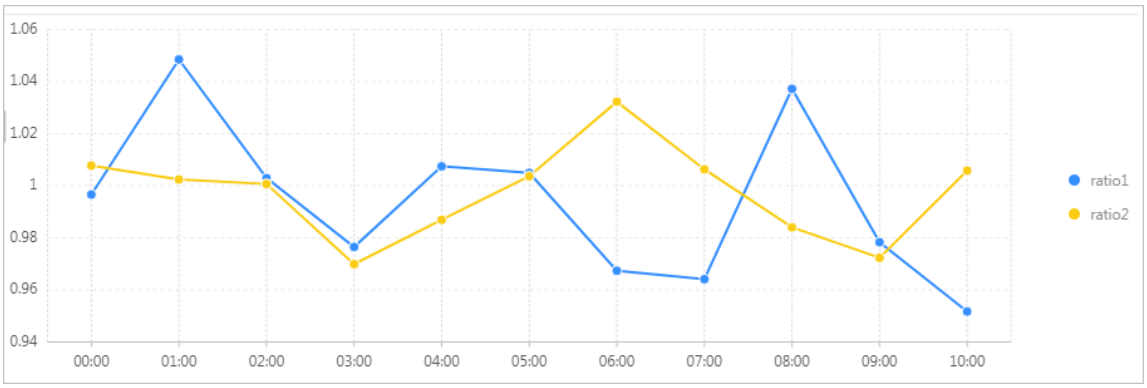
time	diff
00:00	[1176.0,1180.0,1167.0,0.9966101694915255,1.0077120822622108]
01:00	[10077.0,9611.0,10053.0,1.0484861096660077,1.0023873470605789]
02:00	[26921.0,26842.0,26903.0,1.002943148796662,1.0006690703638999]

    - **1176.0** indicates the PVs of the current time period, for example, Dec 25, 2020, 00:00 ~ Dec 25, 2020, 01:00.
    - **1180** indicates the PVs of the same time period the day before, for example, Dec 24, 2020, 00:00 ~ Dec 24, 2020, 01:00.
    - **1167.0** indicates the PVs of the same time period two days before, for example, Dec 23, 2020, 00:00:00 ~ Dec 23, 2020, 01:00:00.
    - **0.9966101694915255** indicates the ratio of the PVs of the current time period to the PVs of the same time period the day before.
    - **1.0077120822622108** indicates the ratio of the PVs of the current period to the PVs of the same period two days before.

- To display the query and analysis result in multiple columns, execute the following query statement:
  - Query statement

```
* |
SELECT
  time,
  diff [1] AS day1,
  diff [2] AS day2,
  diff [3] AS day3,
  diff [4] AS ratio1,
  diff [5] AS ratio2
FROM  (
    SELECT
      time,
      compare(PV, 86400, 172800) as diff
    FROM      (
        SELECT
          count(*) as PV,
          date_format(from_unixtime(__time__), '%H:00') as time
        FROM          log
        GROUP BY
          time
      )
    GROUP BY
      time
    ORDER BY
      time
  )
```

  - Query and analysis result



- Example 3: Calculate the ratio of the PVs of December to the PVs of November in the same year.

  Set the time range to **This Month(Time Frame)** and execute the following query statement. 2592000 indicates the current time minus 2592000 seconds (one month). log indicates the Logstore name. date_trunc('month', __time__) indicates that the date_trunc function is used to truncate a time by month.

- Query statement

```
* |
SELECT
  time,
  compare(PV, 2592000) AS diff
FROM  (
    SELECT
      count(*) AS PV,
      date_trunc('month', __time__) AS time
    FROM      log
    GROUP BY
      time
  )
GROUP BY
  time
ORDER BY
  time
```

- Query and analysis result

time	diff
2021-01-01 00:00:00.000	[11958378.0,448571.0,26.658829928818404]

ts_compare function

The ts_compare function is used to compare the calculation result of the current time period with the calculation result of a time period n seconds before.

> **Notice** The query and analysis results of the ts_compare function must be grouped by the time column by using the GROUP BY clause.

- To compare the calculation result of the current time period with the calculation result of a time period n seconds before, use the following syntax:

```
ts_compare(,n)
```

- To compare the calculation result of the current time period with the calculation results of time periods n1, n2, and n3 seconds before, use the following syntax:

```
ts_compare(,n1,n2,n3...)
```

Parameter	Description
	The value of this parameter is of the double or long type.
n	The time window. Unit: seconds. Example: 3600 (1 hour), 86400 (1 day), 604800 (one week), or 31622400 (one year).

The returned result is a JSON array in the following format: [the current value, the value before n seconds, the ratio of the current value to the value n seconds before, the UNIX timestamp before n seconds].

Calculate the ratio of the PVs of every hour today to the PVs of the previous hour.

Set the time range to **Today(Relative)** and execute the following query statement. 3600 indicates the current time minus 3600 seconds (1 hour). log indicates the Logstore name. date_trunc('hour',__time__) indicates that the date_trunc function is used to truncate a time by hour.

- Query statement

```
* |
SELECT
  time,
  ts_compare(PV, 3600) AS data
FROM(
    SELECT
      date_trunc('hour', __time__) AS time,
      count(*) AS PV
    FROM      log
    GROUP BY
      time
    ORDER BY
      time
  )
GROUP BY
  time
```

- Query and analysis result

time	data
2021-01-27 00:00:00.000	[1160.0,10034.0,0.11560693641618497,1611673200.0]
2021-01-27 01:00:00.000	[10177.0,1160.0,8.773275862068966,1611676800.0]
2021-01-27 02:00:00.000	[26804.0,10177.0,2.6337820575808195,1611680400.0]

8.1.8. Array functions and operators

This topic describes the syntax of array functions and operators. This topic also provides examples on how to use the functions and operators.

The following table describes the array functions and operators that are supported by Log Service.

Notice

Function	Syntax	Description
Subscript operator	[]	Returns the element whose index is from an array. This operator is equivalent to the element_at function.
array_agg function	array_agg()	Returns an array that is created from all values in the column.
array_distinct function	array_distinct()	Removes duplicate elements from an array.
array_except function	array_except(,)	Returns the difference of two arrays.
array_intersect function	array_intersect(,)	Returns the intersection of two arrays.

Function	Syntax	Description
array_join function	array_join(, *delimiter*)	Concatenates the elements of an array into a string by using a specified delimiter. If the array contains a null element, the null element is ignored. **Notice** The array_join function can return a maximum of 1 KB of data. If the size of the returned data exceeds 1 KB, the excess data is truncated.
	array_join(, *delimiter*, *null_replacement*)	Concatenates the elements of an array into a string by using a specified delimiter. If the array contains a null element, the null element is replaced by the value of the *null_replacement* parameter. **Notice** The array_join function can return a maximum of 1 KB of data. If the size of the returned data exceeds 1 KB, the excess data is truncated.
array_max function	array_max()	Returns the maximum value in an array.
array_min function	array_min()	Returns the minimum value in an array.
array_position function	array_position(, *element*)	Returns the index of a specified element in an array. The index starts from 1. If the specified element does not exist, the function returns 0.
array_remove function	array_remove(, *element*)	Removes a specified element from an array.
array_sort function	array_sort()	Sorts the elements in an array in ascending order. If the array contains a null element, the null element is placed at the end.
array_transpose function	array_transpose()	Transposes a matrix and returns a new two-dimensional array that is created from the elements in the matrix. The elements are located by using the same indexes.
array_union function	array_union(,)	Returns the union of two arrays.
cardinality function	cardinality()	Returns the number of elements in an array.
concat function	concat(, ...)	Concatenates multiple arrays into one array.

Function	Syntax	Description
contains function	contains(, *element*)	Checks whether an array contains a specified element. If the array contains the specified element, the function returns true.
element_at function	element_at(,)	Returns the element whose index is from an array.
filter function	filter(, *lambda_expression*)	Filters elements in an array based on a lambda expression and returns elements that match the lambda expression.
flatten function	flatten()	Transforms a two-dimensional array into a one-dimensional array.
reduce function	reduce(, *lambda_expression*)	Returns the sum of the elements in an array based on a lambda expression.
reverse function	reverse()	Reverses the elements in an array.
sequence function	sequence(,)	Returns an array of elements within a specified range. The elements are consecutive and incremental. The default incremental step is 1.
	sequence(, , *step*)	Returns an array of elements within a specified range. The elements are consecutive and incremental. The incremental step is a custom value.
shuffle function	shuffle()	Shuffles the elements in an array.
slice function	slice(, *start*, *length*)	Returns a subset of an array.
transform function	transform(, *lambda_expression*)	Transforms each element in an array by using a lambda expression.
zip function	zip(, ...)	Merges multiple arrays into a two-dimensional array. Elements that have the same index in the input arrays form a new array in the two-dimensional array.
zip_with function	zip_with(, , *lambda_expression*)	Merges two arrays into a single array by using a lambda expression.

Subscript operator

The subscript operator is used to return the element whose index is from an array. This operator is equivalent to the element_at function.

```
[]
```

Parameter	Description
	The index of an element in an array. The index starts from 1. The value of this parameter is of the bigint type.

The data type of the specified element.

Obtain the first element from the value of the number field.

- Sample field

```
number:[49,50,45,47,50]
```

- Query statement

```
* | SELECT cast(json_parse(number) as array(bigint)) [1]
```

- Query and analysis result

array_agg function

The array_agg function is used to return an array that is created from all values in the column.

```
array_agg ()
```

Parameter	Description
	The value of this parameter is of an arbitrary data type.

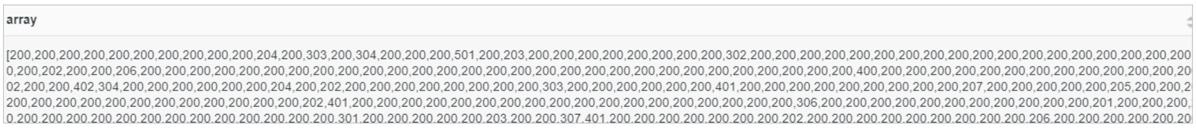
The array type.

Obtain an array that is created from all values in the status column.

- Query statement

```
* | SELECT array_agg(status) AS array
```

- Query and analysis result



array_distinct function

The array_distinct function is used to remove duplicate elements from an array.

```
array_distinct()
```

Parameter	Description
	The value of this parameter is of the array type.

The array type.

Remove duplicate elements from the value of the number field.

- Sample field

```
number:[49,50,45,47,50]
```

- Query statement

```
*| SELECT array_distinct(cast(json_parse(number) as array(bigint)))
```

- Query and analysis result

array_except function

The array_except function is used to return the difference of two arrays.

```
array_except(, )
```

Parameter	Description
	The value of this parameter is of the array type.
	The value of this parameter is of the array type.

The array type.

Obtain the difference of the [1,2,3,4,5] and [1,3,5,7] arrays.

- Query statement

```
* | SELECT array_except(array[1,2,3,4,5],array[1,3,5,7])
```

- Query and analysis result

array_intersect function

The array_intersect function is used to return the intersection of two arrays.

```
array_intersect(, )
```

Parameter	Description
	The value of this parameter is of the array type.
	The value of this parameter is of the array type.

The array type.

Obtain the intersection of the [1,2,3,4,5] and [1,3,5,7] arrays.

- Query statement

```
* | SELECT array_intersect(array[1,2,3,4,5],array[1,3,5,7])
```

- Query and analysis result

array_join function

The array_join function is used to concatenate the elements of an array into a string by using a specified delimiter.

- The following syntax of the array_join function is used to concatenate the elements of an array into a string by using a specified delimiter. If the array contains a null element, the null element is ignored.

```
array_join(, delimiter)
```

- The following syntax of the array_join function is used to concatenate the elements of an array into a string by using a specified delimiter. If the array contains a null element, the null element is replaced by the value of the *null_replacement* parameter.

```
array_join(, delimiter,null_replacement)
```

Parameter	Description
	The value of this parameter is of an arbitrary array type.
delimiter	The delimiter that is used to connect elements. You can specify a string for this parameter.
null_replacement	The string that is used to replace a null element.

The varchar type.

Concatenate the elements of the [null, 'Log','Service'] array into a string by using space characters and replace the null element with Alicloud.

- Query statement

```
* | SELECT array_join(array[null,'Log','Service'],' ','Alicloud')
```

- Query and analysis result

array_max function

The array_max function is used to return the maximum value in an array.

```
array_max()
```

Parameter	Description
	The value of this parameter is of the array type. **Notice** If an array contains a null element, the function returns null.

The data type of elements in the parameter value.

Obtain the maximum value in an array.

- Sample field

```
number:[49,50,45,47,50]
```

- Query statement

```
*| SELECT array_max(try_cast(json_parse(number) as array(bigint))) AS max_number
```

- Query and analysis result

array_min function

The array_min function is used to return the minimum value in an array.

```
array_min()
```

Parameter	Description
	The value of this parameter is of the array type. **Notice** If an array contains a null element, the function returns null.

The data type of elements in the parameter value.

Obtain the minimum value in an array.

- Sample field

```
number:[49,50,45,47,50]
```

- Query statement

```
*| SELECT array_min(try_cast(json_parse(number) as array(bigint))) AS min_number
```

- Query and analysis result

array_position function

The array_position function is used to return the index of a specified element in an array. The index starts from 1. If the specified element does not exist, the function returns 0.

```
array_position(, element)
```

Parameter	Description
	The value of this parameter is of the array type.

Parameter	Description
element	The value of this parameter is the element whose index you want to obtain. Note If the element is null, the function returns null.

The bigint type.

Obtain the index of 45 from the [49,45,47] array.

- Query statement

```
* | SELECT array_position(array[49,45,47],45)
```

- Query and analysis result

array_remove function

The array_remove function is used to remove a specified element from an array.

```
array_remove(, element)
```

Parameter	Description
	The value of this parameter is of the array type.
element	The value of this parameter is the element that you want to remove. Note If the element is null, the function returns null.

The array type.

Remove 45 from the [49,45,47] array.

- Query statement

```
* | SELECT array_remove(array[49,45,47],45)
```

- Query and analysis result

array_sort function

The array_sort function is used to sort the elements in an array in ascending order. If the array contains a null element, the null element is placed at the end.

```
array_sort()
```

Parameter	Description
	The value of this parameter is of the array type.

The array type.

Sort the elements in the ['b', 'd', null, 'c', 'a'] array in ascending order.

- Query statement

```
* | SELECT array_sort(array['b','d',null,'c','a'])
```

- Query and analysis result

array_transpose function

The array_transpose function is used to transpose a matrix and return a new two-dimensional array that is created from the elements in the matrix. The elements are located by using the same indexes.

```
array_transpose()
```

Parameter	Description
	The value of this parameter is of the array(double) type.

The array(double) type.

Create a two-dimensional array from elements that are located by using the same indexes in a different two-dimensional array. For example, in the [0,1,2,3], [10,19,18,17], and [0,9,8,7] arrays, 0, 10, and 9 are all located by using the index 1. This way, the new array [0.0,10.0,9.0] is formed.

- Query statement

```
* | SELECT array_transpose(array[array[0,1,2,3],array[10,19,18,17],array[9,8,7]])
```

- Query and analysis result

array_union function

The array_union function is used to return the union of two arrays.

```
array_union(, )
```

Parameter	Description
	The value of this parameter is of the array type.
	The value of this parameter is of the array type.

The array type.

Obtain the union of the [1,2,3,4,5] and [1,3,5,7] arrays.

- Query statement

```
* | SELECT array_union(array[1,2,3,4,5],array[1,3,5,7])
```

- Query and analysis result

cardinality function

The cardinality function is used to return the number of elements in an array.

```
cardinality()
```

Parameter	Description
	The value of this parameter is of the array type.

The bigint type.

Obtain the number of elements in the value of the number field.

- Sample field

```
number:[49,50,45,47,50]
```

- Query statement

```
*| SELECT cardinality(cast(json_parse(number) as array(bigint)))
```

- Query and analysis result

concat function

The concat function is used to concatenate multiple arrays into one array.

```
concat(, …)
```

Parameter	Description
	The value of this parameter is of the array type.
	The value of this parameter is of the array type.

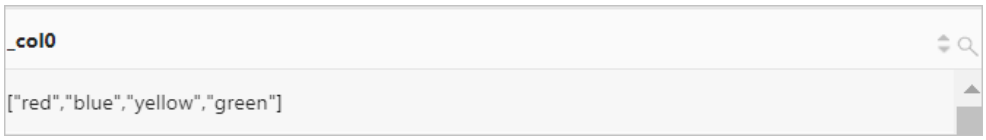
The array type.

Concatenate the ['red','blue'] and ['yellow','green'] arrays into one array.

- Query statement

```
* | SELECT concat(array['red','blue'],array['yellow','green'])
```

- Query and analysis result



contains function

The contains function is used to check whether an array contains a specified element. If the array contains the specified element, the function returns true.

```
contains(, element)
```

Parameter	Description
	The value of this parameter is of the array type.
element	The value of this parameter is the element that you want to check.

The Boolean type.

Check whether the value of the region field contains cn-beijing.

- Sample field

```
region:["cn-hangzhou","cn-shanghai","cn-beijing"]
```

- Query statement

```
*| SELECT contains(cast(json_parse(region) as array(varchar)),'cn-beijing')
```

- Query and analysis result

element_at function

The element_at function is used to return the element whose index is from an array.

```
element_at(, )
```

Parameter	Description
	The value of this parameter is of the array type.
	The index of an element in an array. The index starts from 1. The value of this parameter is of the bigint type.

An arbitrary data type.

Obtain the second element from the value of the number field.

- Sample field

```
number:[49,50,45,47,50]
```

- Query statement

```
* |
SELECT
  element_at(cast(json_parse(number) AS array(varchar)), 2)
```

- Query and analysis result

filter function

The filter function is used to filter elements in an array based on a lambda expression and return elements that match the lambda expression.

```
filter(, lambda_expression)
```

Parameter	Description
	The value of this parameter is of the array type.
lambda_expression	The lambda expression. For more information, see Lambda expressions.

The array type.

Obtain the elements that are greater than 0 from the [5,-6,null,7] array by using the lambda expression `x -> x > 0`.

- Query statement

```
* | SELECT filter(array[5,-6,null,7],x -> x > 0)
```

- Query and analysis result

flatten function

The flatten function is used to transform a two-dimensional array into a one-dimensional array.

```
flatten()
```

Parameter	Description
	The value of this parameter is of the array type.

The array type.

Transform the two-dimensional array [array[1,2,3,4],array[5,2,2,4] into a one-dimensional array.

- Query statement

```
* | SELECT flatten(array[array[1,2,3,4],array[5,2,2,4]])
```

- Query and analysis result

reduce function

The reduce function is used to return the sum of the elements in an array based on a lambda expression.

```
reduce(, lambda_expression)
```

Parameter	Description
	The value of this parameter is of the array type.
lambda_expression	The lambda expression. For more information, see Lambda expressions.

The bigint type.

Obtain the sum of the elements in the [5,20,50] array.

- Query statement

```
* | SELECT reduce(array[5,20,50],0,(s, x) -> s + x, s -> s)
```

- Query and analysis result

reverse function

The reverse function is used to reverse the elements in an array.

```
reverse()
```

Parameter	Description
	The value of this parameter is of the array type.

The array type.

Reverse the elements in the [1,2,3,4,5] array.

- Query statement

```
* | SELECT reverse(array[1,2,3,4,5])
```

- Query and analysis result

sequence function

The sequence function is used to return an array of elements within a specified range. The elements are consecutive and incremental.

- The following syntax of the sequence function uses the default incremental step. The default step is 1.

```
sequence(, )
```

- The following syntax of the sequence function uses a custom incremental step:

```
sequence(, , step)
```

Parameter	Description
	The value of this parameter is of the bigint or timestamp type. UNIX timestamps and date and time expressions are supported.
	The value of this parameter is of the bigint or timestamp type. UNIX timestamps and date and time expressions are supported.
step	The incremental step. If the values of the x and y parameters are date and time expressions, the value of the *step* parameter is in one of the following formats: • `interval ' n' year to month` : The incremental step is n years. • `interval 'n' day to second` : The incremental step is n days.

The array type.

- Example 1: Obtain the even numbers within the range from 0 to 10.
  - Query statement

```
* | SELECT sequence(0,10,2)
```

  - Query and analysis result

- Example 2: Obtain the dates within the range from 2017-10-23 to 2021-08-12 at the incremental step of 1 year.
  - Query statement

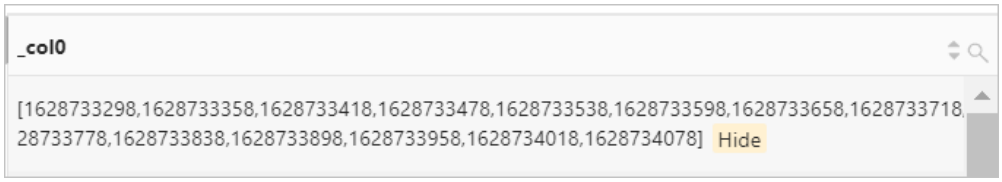
```
ww* | SELECT  sequence(from_unixtime(1508737026),from_unixtime(1628734085),interval '1' year to month )
```

  - Query and analysis result

- Example 3: Obtain the UNIX timestamps within the range from 1628733298 to 1628734085 at the incremental step of 60 seconds.
  - Query statement

```
* | SELECT  sequence(1628733298,1628734085,60)
```

- Query and analysis result



shuffle function

The shuffle function is used to shuffle the elements in an array.

```
shuffle()
```

Parameter	Description
	The value of this parameter is of the array type.

The array type.

Shuffle the elements in the [1,2,3,4,5] array.

- Query statement

```
*| SELECT shuffle(array[1,2,3,4,5])
```

- Query and analysis result

slice function

The slice function is used to return a subset of an array.

```
slice(, start, length)
```

Parameter	Description
	The value of this parameter is of the array type.
start	The index at which Log Service starts to extract elements. • If the value of the *start* parameter is negative, Log Service starts to extract elements from the end of the array. • If the value of the *start* parameter is a positive number, Log Service starts to extract elements from the beginning of the array.
length	The number of elements that you want to include in the subset.

The array type.

Obtain a subset of the [1,2,4,5,6,7,7] array from the third element with two elements.

- Query statement

```
* | SELECT slice(array[1,2,4,5,6,7,7],3,2)
```

- Query and analysis result

transform function

The transform function is used to transform each element in an array by using a lambda expression.

```
transform(, lambda_expression)
```

Parameter	Description
	The value of this parameter is of the array type.
lambda_expression	The lambda expression. For more information, see Lambda expressions.

The array type.

Add 1 to each element in the [5,6] array and return a new array.

- Query statement

```
* | SELECT transform(array[5,6],x -> x + 1)
```

- Query and analysis result

zip function

The zip function is used to merge multiple arrays into a two-dimensional array. Elements that have the same index in the input arrays form a new array in the two-dimensional array.

```
zip(, ...)
```

Parameter	Description
	The value of this parameter is of the array type.
	The value of this parameter is of the array type.

The array type.

Merge the [1, 2,3], ['1b', null, '3b'], and [1, 2,3] arrays into a two-dimensional array.

- Query statement

```
* | SELECT zip(array[1,2,3], array['1b',null,'3b'],array[1,2,3])
```

- Query and analysis result

zip_with function

The zip_with function is used to merge two arrays into a single array by using a lambda expression.

```
zip_with(, , lambda_expression)
```

Parameter	Description
	The value of this parameter is of the array type.
	The value of this parameter is of the array type.
lambda_expression	The lambda expression. For more information, see Lambda expressions.

The array type.

Use the lambda expression `(x, y) -> x + y` to add the corresponding elements in the [1, 2] and [3, 4] arrays and return a new array.

- Query statement

```
SELECT zip_with(array[1,2], array[3,4],(x,y) -> x + y)
```

- Query and analysis result

8.1.9. Map functions and operators

This topic describes the syntax of map functions and operators. This topic also provides examples on how to use the functions and operators.

The following table describes the map functions and operators that are supported by Log Service.

Notice

Function	Syntax	Description
Subscript operator	[]	Is used to retrieve the value of a key from a map.
cardinality function	cardinality()	Returns the size of a map.
element_at function	element_at(, *key*)	Returns the value of a key in a map.
histogram function	histogram()	Groups query and analysis results and returns data in the JSON format.
histogram_u function	histogram_u()	Groups query and analysis results and returns data in multiple rows and multiple columns.

Function	Syntax	Description
map() function	map()	Returns an empty map.
map function	map(,)	Returns a map that is created by using two arrays.
map_agg function	map_agg(,)	Returns a map that is created by using and . is the key in the map. is the value of the key in the map. If has multiple values, a random value is extracted as the value of the key.
map_concat function	map_concat(,...)	Returns the union of multiple maps.
map_filter function	map_filtermap_filter(,*lambda_expression*)	Filters elements in a map based on a lambda expression and returns a new map.
map_keys function	map_keys()	Returns an array that consists of all the keys of a map.
map_values function	map_values()	Returns an array that consists of all the values of a map.
multimap_agg function	multimap_agg(,)	Returns a multimap that is created by using and . is the key in the multimap. is the value of the key in the multimap, and the value is of the array type. If has multiple values, all the values are extracted as the values of the key.

Subscript operator

The subscript operator is used to retrieve the value of a key from a map.

```
[]
```

Parameter	Description
	The value of this parameter is of the varchar type.

An arbitrary data type.

In a log of a data transformation task, the etl_context field has a value of the map type. You can use the subscript operator to retrieve the value of the project key from the value of the etl_context field.

- Sample field

```
etl_context: {
 project:"datalab-148****6461-cn-chengdu"
 logstore:"internal-etl-log"
 consumer_group:"etl-83****4d1965"
 consumer:"etl-b2d40ed****c8d6-291294"
 shard_id:"0" }
```

- Query statement

```
* | SELECT try_cast(json_parse(etl_context) AS map(varchar, varchar))['project']
```

- Query and analysis results

cardinality function

The cardinality function returns the size of a map.

```
cardinality()
```

Parameter	Description
	The value of this parameter is of the map type.

The bigint type.

Use the histogram function to obtain the number of requests for each request method. Then, use the cardinality function to obtain the number of request methods.

- Query statement

```
* |
SELECT
  histogram(request_method) AS request_method,
  cardinality(histogram(request_method)) AS "kinds"
```

- Query and analysis results

request_method	kinds
{"DELETE":5,"POST":7,"GET":41,"PUT":4}	4

element_at function

The element_at function returns the value of a key in a map.

```
element_at(, key)
```

Parameter	Description
	The value of this parameter is of the map type.
key	The value of this parameter is a key in a map.

An arbitrary data type.

Use the histogram function to obtain the number of requests for each request method. Then, use the element_at function to obtain the value of the DELETE field.

- Query statement

```
* |
SELECT
  histogram(request_method) AS request_method,
  element_at(histogram(request_method),'DELETE') AS "count"
```

- Query and analysis results

request_method	count
{"HEAD":9,"DELETE":140,"POST":319,"GET":1298,"PUT":337}	140

histogram function

The histogram function groups query and analysis results and returns data in the JSON format. This function is equivalent to `* | SELECT count(*) GROUP BY` .

```
histogram()
```

Parameter	Description
	The value of this parameter is of an arbitrary data type.

The map type.

Use the histogram function to obtain the number of requests for each request method.

- Query statement

```
* | SELECT histogram(request_method) AS request_method
```

- Query and analysis results

request_method
{"HEAD":30,"DELETE":564,"POST":1382,"GET":5420,"PUT":1334}

histogram_u function

The histogram_u function groups query and analysis results and returns data in multiple rows and multiple columns.

```
histogram_u()
```

Parameter	Description
	The value of this parameter is of an arbitrary data type.

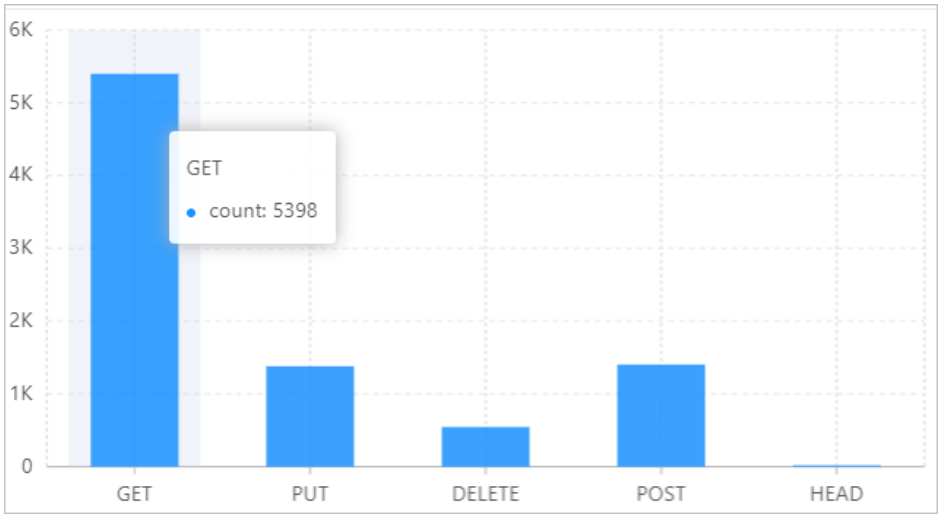
The bigint type.

Use the histogram_u function to obtain the number of requests for each request method and then display the numbers in a column chart.

- Query statement

```
*|SELECT  histogram_u(request_method) as request_method
```

- Query and analysis results



map() function

The map() function returns an empty map.

```
map()
```

The map type.

Use the map() function to obtain an empty map.

- Query statement

```
*| SELECT map()
```

- Query and analysis results

map function

The map function returns a map that is created by using two arrays.

```
map(,)
```

Parameter	Description
	The value of this parameter is of the array type.
	The value of this parameter is of the array type.

The map type.

The class field specifies classes. The number field specifies the numbers of students in the classes. The values of the two fields are of the array type. Use the map function to create a map based on the values of the two fields. In the returned result, each class is mapped to the number of students in the class.

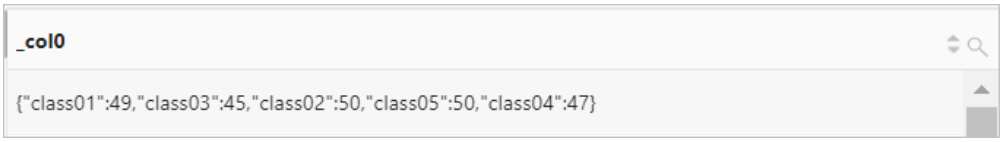
- Sample field

```
class:["class01","class02","class03","class04","class05"]
number:[49,50,45,47,50]
```

- Query statement

```
* | SELECT map(try_cast(json_parse(class) AS array(varchar)) ,try_cast(json_parse(number) AS array
(bigint)))
```

- Query and analysis results



map_agg function

The map_agg function returns a map that is created by using and . is the key in the map. is the value of the key in the map. If has multiple values, a random value is extracted as the value of the key.

```
map_agg(,)
```

Parameter	Description
	The value of this parameter is of an arbitrary data type.
	The value of this parameter is of an arbitrary data type.

The map type.

Extract the values of the request_method and request_time fields and then use the extracted values to create a map. The value of request_method is the key in the map. The value of request_time is the value of the key in the map.

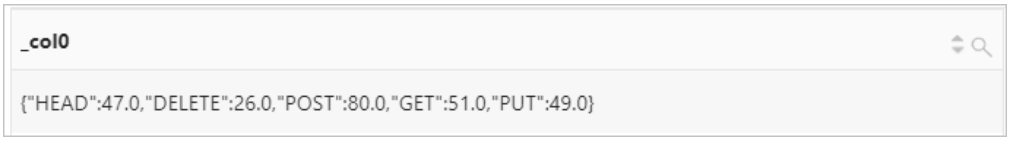
- Sample field

```
request_method:POST
request_time:80
```

- Query statement

```
* | SELECT map_agg(request_method,request_time)
```

- Query and analysis results



map_concat function

The map_concat function returns the union of multiple maps.

```
map_concat(,)
```

Parameter	Description
	The value of this parameter is of the map type.
	The value of this parameter is of the map type.

The map type.

In a log of a data transformation task, the etl_context and progress fields have values of the map type. You can use the map_concat function to obtain the union of the values.

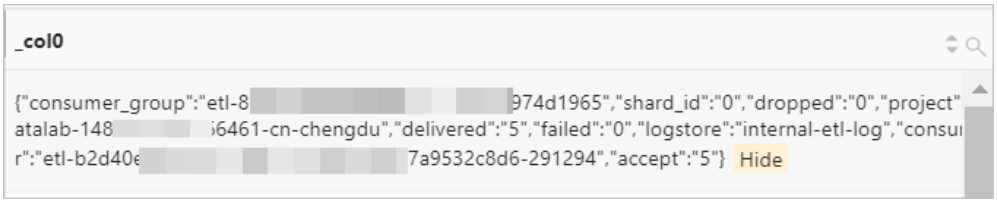
- Sample field

```
etl_context: {
 project:"datalab-148****6461-cn-chengdu"
 logstore:"internal-etl-log"
 consumer_group:"etl-83****4d1965"
 consumer:"etl-b2d40ed****c8d6-291294"
 shard_id:"0" }
progress: {
 accept:3
 dropped:0
 delivered:3
 failed:0 }
```

- Query statement

```
* |
SELECT
  map_concat(
    cast (
      json_parse(etl_context) AS map(varchar, varchar)
    ),
    cast (json_parse(progress) AS map(varchar, varchar))
  )
```

- Query and analysis results



map_filter function

The map_filter function filters elements in a map based on a lambda expression and returns a new map.

```
map_filter(,lambda_expression)
```

Parameter	Description
	The value of this parameter is of the map type.
lambda_expression_expression	The value of this parameter is a lambda expression. For more information, see Lambda expressions.

The map type.

Create a map that does not contain null values from two arrays by using the lambda expression `(k, v) -> v is not null` .

- Query statement

```
* | SELECT map_filter(map(array[10, 20, 30], array['a', NULL, 'c']), (k, v) -> v is not null)
```

- Query and analysis results

map_keys function

The map_keys function returns an array that consists of all the keys of a map.

```
map_keys()
```

Parameter	Description
	The value of this parameter is of the map type.

The array type.

In a log of a data transformation task, the etl_context field has a value of the map type. You can use the map_keys function to obtain all the keys in the value of the etl_context field.

- Sample field

```
etl_context: {
 project:"datalab-148****6461-cn-chengdu"
 logstore:"internal-etl-log"
 consumer_group:"etl-83****4d1965"
 consumer:"etl-b2d40ed****c8d6-291294"
 shard_id:"0" }
```

- Query statement

```
* | SELECT map_keys(try_cast(json_parse(etl_context) AS map(varchar, varchar)))
```

- Query and analysis results

map_values function

The map_values function returns an array that consists of all the values of a map.

```
map_values()
```

Parameter	Description
	The value of this parameter is of the map type.

The array type.

In a log of a data transformation task, the etl_context field has a value of the map type. You can use the map_values function to obtain all the values of keys in the value of the etl_context field.

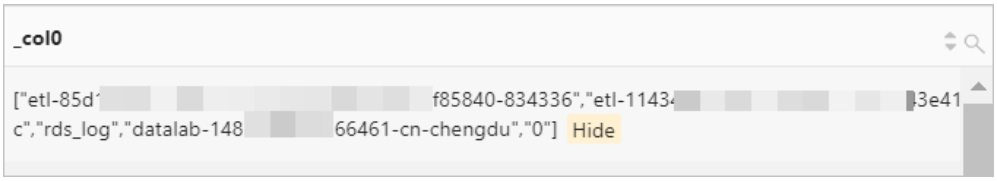
- Sample field

```
etl_context: {
 project:"datalab-148****6461-cn-chengdu"
 logstore:"internal-etl-log"
 consumer_group:"etl-83****4d1965"
 consumer:"etl-b2d40ed****c8d6-291294"
 shard_id:"0" }
```

- Query statement

```
* | SELECT map_values(try_cast(json_parse(etl_context) AS map(varchar, varchar)))
```

- Query and analysis results



multimap_agg function

The multimap_agg function returns a multimap that is created by using and . is the key in the multimap. is the value of the key in the multimap, and the value is of the array type. If has multiple values, all the values are extracted as the values of the key.

```
multimap_agg(,)
```

Parameter	Description
	The value of this parameter is of an arbitrary data type.
	The value of this parameter is of an arbitrary data type.

The map type.

Extract all the values of the request_method and request_time fields and then use the extracted values to create a multimap. The value of request_method is the key in the multimap. The value of request_time is the value of the key in the multimap, and the value of the key is of the array type.

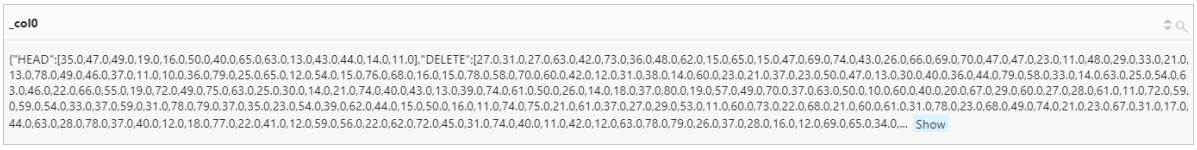
- Sample field

```
request_method:POST
request_time:80
```

- Query statement

```
* | SELECT multimap_agg(request_method,request_time)
```

- Query and analysis results



8.1.10. Mathematical calculation functions

This topic describes the syntax of mathematical calculation functions. This topic also provides examples on how to use the functions.

Note

- The following operators are supported:

  + - * / %

- If you want to use strings in analytic statements, you must enclose the strings in single quotation marks (''). Strings that are not enclosed or are enclosed in double quotation marks ("") are considered field names or column names. For example, 'status' is considered the status string, and status or "status" is considered a log field whose name is status.

Function	Syntax	Description
abs function	abs()	Calculates the absolute value of .
acos function	acos()	Calculates the arc cosine of .
asin function	asin()	Calculates the arc sine of .
atan function	atan()	Calculates the arc tangent of .
atan2 function	atan2(,)	Calculates the arc tangent of divided by .
cbrt function	cbrt()	Calculates the cube root of .
ceil function	ceil()	Rounds up to the nearest integer. The ceil function is an alias of the ceiling function.
ceiling function	ceiling()	Rounds up to the nearest integer.
cos function	cos()	Calculates the cosine of .
cosh function	cosh()	Calculates the hyperbolic cosine of .
cosine_similarity function	cosine_similarity(,)	Calculates the cosine similarity between and .
degrees function	degrees()	Converts an angle in radians to its equivalent in degrees.
e function	e()	Returns the value of e, which is the base of the natural logarithm.
exp function	exp()	Raises e to the power of .
floor function	floor()	Rounds down to the nearest integer.
from_base function	from_base(,)	Converts to a base number.
ln function	ln()	Calculates the natural logarithm of .
infinity function	infinity()	Returns a value that represents positive infinity.
is_nan function	is_nan()	Determines whether is Not a Number (NaN).

Function	Syntax	Description
log2 function	log2()	Calculates the base-2 logarithm of .
log10 function	log10()	Calculates the base-10 logarithm of .
log function	log(,)	Calculates the base- logarithm of .
mod function	mod(,)	Calculates the remainder of divided by .
nan function	nan()	Returns a value that is NaN.
pi function	pi()	Returns the value of π to 15 decimal places.
pow function	pow(,)	Raises to the power of . The pow function is an alias of the power function.
power function	power(,)	Raises to the power of .
radians function	radians()	Converts an angle in degrees to its equivalent in radians.
rand function	rand()	Returns a random number.
random function	random()	Returns a random number in the range [0,1).
	random()	Returns a random number in the range [0,x).
round function	round()	Rounds to the nearest integer.
	round(, *n*)	Rounds to the nearest decimal with n decimal places.
sign function	sign()	Returns the sign of . Valid values: 1, 0, and -1.
sin function	sin()	Calculates the sine of .
sqrt function	sqrt()	Calculates the square root of .
tan function	tan()	Calculates the tangent of .
tanh function	tanh()	Calculates the hyperbolic tangent of .
to_base function	to_base(,)	Converts to a base string.
truncate function	truncate()	Removes the fractional part of .
width_bucket function	width_bucke(,*bound1*,*bound2*,*numBuckets*)	Divides a numeric range into buckets of equal width and returns the bucket number of .
	width_bucke(,*bins*)	Returns the bucket number of in the range of buckets that are specified by an array.

abs function

The abs function calculates the absolute value of .

```
abs()
```

Parameter	Description
	The value of this parameter is of the smallint, integer, real, tinyint, bigint, double, or decimal type.

Same as the data type of the parameter value.

Calculate the absolute value of -25.

- Query statement

```
* | select abs(-25)
```

- Query and analysis results

acos function

The acos function calculates the arc cosine of .

```
acos()
```

Parameter	Description
	The value of this parameter is of the double type. Valid values: [-1,1]. If the value is out of the range [-1,1], the function returns NaN.

The double type.

Calculate the arc cosine of the 45° angle.

- Query statement

```
* | SELECT acos(pi()/4)
```

- Query and analysis results

asin function

The asin function calculates the arc sine of .

```
asin()
```

Parameter	Description
	The value of this parameter is of the double type. Valid values: [-1,1]. If the value is out of the range [-1,1], the function returns NaN.

The double type.

Calculate the arc sine of the 45° angle.

- Query statement

```
* | SELECT  asin(pi()/4)
```

- Query and analysis results

_col0
0.9033391107665127

atan function

The atan function calculates the arc tangent of .

```
atan()
```

Parameter	Description
	The value of this parameter is of the double type.

The double type.

Calculate the arc tangent of the 45° angle.

- Query statement

```
* | SELECT atan(pi()/4)
```

- Query and analysis results

_col0
0.6657737500283538

atan2 function

The atan2 function calculates the arc tangent of divided by .

```
atan2(,)
```

Parameter	Description
	The value of this parameter is of the double type.
	The value of this parameter is of the double type.

The double type.

Calculate the arc tangent of the 30° angle.

- Query statement

```
* | SELECT atan2(pi(),6)
```

- Query and analysis results

cbrt function

The cbrt function calculates the cube root of .

```
cbrt()
```

Parameter	Description
	The value of this parameter is of the double type.

The double type.

Calculate the cube root of 100.

- Query statement

```
* | select cbrt(100)
```

- Query and analysis results

ceil function

The ceil function rounds up to the nearest integer. The ceil function is an alias of the ceiling function.

```
ceil()
```

Parameter	Description
	The value of this parameter is of the tinyint, smallint, integer, real, bigint, double, or decimal type. • If the value of is a positive number, the function rounds the value away from 0. • If the value of is a negative number, the function rounds the value towards 0.

Same as the data type of the parameter value.

Round the value of the request_time field up to the nearest integer.

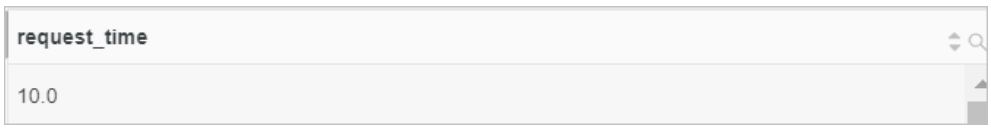
- Sample field

```
request_time:9.3
```

- Query statement

```
* | SELECT ceil(request_time) AS request_time
```

- Query and analysis results



ceiling function

The ceiling function rounds up to the nearest integer.

```
ceiling()
```

Parameter	Description
	The value of this parameter is of the tinyint, smallint, integer, real, bigint, double, or decimal type. • If the value of is a positive number, the function rounds the value away from 0. • If the value of is a negative number, the function rounds the value towards 0.

Same as the data type of the parameter value.

Round the value of the request_time field up to the nearest integer.

- Sample field

```
request_time:9.3
```

- Query statement

```
* | SELECT ceiling(request_time) AS request_time
```

- Query and analysis results

cos function

The cos function calculates the cosine of .

```
cos()
```

Parameter	Description
	The value of this parameter is of the double type.

The double type.

Calculate the cosine of the 30° angle.

- Query statement

```
* | SELECT cos(pi()/6)
```

- Query and analysis results

cosh function

The cosh function calculates the hyperbolic cosine of .

```
cosh()
```

Parameter	Description
	The value of this parameter is of the double type.

The double type.

Calculate the hyperbolic cosine of the 30° angle.

- Query statement

```
* | SELECT cosh(pi()/6)
```

- Query and analysis results

cosine_similarity function

The cosine_similarity function calculates the cosine similarity between and .

```
cosine_similarity(,)
```

Parameter	Description
	The value of this parameter is of the map(varchar,double) type.
	The value of this parameter is of the map(varchar,double) type.

The double type.

Calculate the cosine similarity between two vectors.

- Query statement

```
* | SELECT cosine_similarity(MAP(ARRAY['a'], ARRAY[1.0]), MAP(ARRAY['a'], ARRAY[2.0]))
```

- Query and analysis results

degrees function

The degrees function converts an angle in radians to its equivalent in degrees.

```
degrees()
```

Parameter	Description
	The value of this parameter is of the double type.

The double type.

Convert π in radians to its equivalent in degrees.

- Query statement

```
* | SELECT degrees(pi())
```

- Query and analysis results

e function

The e function returns the value of e, which is the base of the natural logarithm.

```
e()
```

The double type.

Obtain the value of e.

- Query statement

```
* | SELECT e()
```

- Query and analysis results

exp function

The exp function raises e to the power of x.

```
exp()
```

Parameter	Description
	The value of this parameter is of the double type.

The double type.

Raise e to the power of 3.

- Query statement

```
* | SELECT exp(3)
```

- Query and analysis results

floor function

The floor function rounds down to the nearest integer.

```
floor()
```

Parameter	Description
	The value of this parameter is of the tinyint, smallint, integer, real, bigint, double, or decimal type. • If the value of is a positive number, the function rounds the value towards 0. • If the value of is a negative number, the function rounds the value away from 0.

The double type.

Round the value of the request_time field down to the nearest integer.

- Sample field

  ```
  request_time:10.3
  ```

- Query statement

  ```
  * | SELECT ceiling(request_time) AS request_time
  ```

- Query and analysis results

from_base function

The from_base function converts to a base number.

```
from_base(,)
```

Parameter	Description
	The value of this parameter is of the varchar type.
	The value of this parameter is of the bigint type. The value specifies a numeral system. Valid values: [2,36].

The bigint type.

Convert the string 1101 to a number.

- Query statement

  ```
  * | SELECT  from_base('1101',2)
  ```

- Query and analysis results

ln function

The ln function calculates the natural logarithm of .

```
ln()
```

Parameter	Description
	The value of this parameter is of the double type. The value must be greater than 0.

The double type.

Calculate the natural logarithm of 2.

- Query statement

```
* | SELECT ln(2)
```

- Query and analysis results

infinity function

The infinity function returns a value that represents positive infinity.

```
infinity()
```

The double type.

Obtain a value that represents positive infinity.

- Query statement

```
* | SELECT infinity()
```

- Query and analysis results

is_nan function

The is_nan function determines whether is NaN. If it is, the function returns true.

```
is_nan()
```

Parameter	Description
	The value of this parameter is of the double type.

The Boolean type.

Check whether the value of the status field is NaN.

- Query statement

```
* | SELECT is_nan(status)
```

- Query and analysis results

log2 function

The log2 function calculates the base-2 logarithm of .

```
log2()
```

Parameter	Description
	The value of this parameter is of the double type.

The double type.

Calculate the base-2 logarithm of 100.

- Query statement

```
* | SELECT log2(100)
```

- Query and analysis results

log10 function

The log10 function calculates the base-10 logarithm of .

```
log10()
```

Parameter	Description
	The value of this parameter is of the double type.

The double type.

Calculate the base-10 logarithm of 100.

- Query statement

```
* | SELECT log10(100)
```

- Query and analysis results

log function

The log function calculates the base- logarithm of .

```
log(,)
```

Parameter	Description
	The value of this parameter is of the double type.
	The value of this parameter is of the double type.

The double type.

Calculate the base-5 logarithm of 100.

- Query statement

```
* | SELECT log(100,5)
```

- Query and analysis results

mod function

The mod function calculates the remainder of divided by .

```
mod(,)
```

Parameter	Description
	The value of this parameter is of the tinyint, smallint, integer, real, bigint, double, or decimal type.
	The value of this parameter is of the tinyint, smallint, integer, real, bigint, double, or decimal type.

Same as the data type of the parameter values.

Calculate the remainder of 100 divided by 30.

- Query statement

```
* | SELECT mod(100,30)
```

- Query and analysis results

nan function

The nan function returns a value that is NaN.

```
nan()
```

The double type.

Obtain a value that is NaN.

- Query statement

```
* | SELECT nan()
```

- Query and analysis results

pi function

The pi function returns the value of π to 15 decimal places.

```
pi()
```

The double type.

Obtain the value of π to 15 decimal places.

- Query statement

```
* | SELECT pi()
```

- Query and analysis results

pow function

The pow function raises to the power of . The pow function is an alias of the power function.

```
pow(,)
```

Parameter	Description
	The value of this parameter is of the double type.
	The value of this parameter is of the double type.

The double type.

Raise 2 to the power of 5.

- Query statement

```
* | SELECT pow(2,5)
```

- Query and analysis results

power function

The power function raises to the power of .

```
power(,)
```

Parameter	Description
	The value of this parameter is of the double type.
	The value of this parameter is of the double type.

The double type.

Raise 2 to the power of 5.

- Query statement

```
* | SELECT power(2,5)
```

- Query and analysis results

radians function

The radians function converts an angle in degrees to its equivalent in radians.

```
radians()
```

Parameter	Description
	The value of this parameter is of the double type.

The double type.

Convert the 180° angle in degrees to its equivalent in radians.

- Query statement

```
* | SELECT radians(180)
```

- Query and analysis results

rand function

The rand function returns a random number.

```
rand()
```

The double type.

Obtain a random number.

- Query statement

```
* | select rand()
```

- Query and analysis results

_col0
0.8742241064002435

random function

The random function returns a random number in the range [0,x).

- The following random function returns a random number in the range [0,1).

```
random()
```

- The following random function returns a random number in the range [0,x).

```
random()
```

Parameter	Description
	The value of this parameter is of the tinyint, smallint, integer, or bigint type.

Same as the data type of the parameter value.

Obtain a random number in the range [0,100).

- Query statement

```
* | select random(100)
```

- Query and analysis results

_col0
44

round function

The round function rounds to the nearest integer or decimal. If *n* is specified, the function retains n decimal places. If *n* is not specified, the function rounds to the nearest integer.

- The following round function rounds to the nearest integer.

```
round()
```

- The following round function rounds to the nearest decimal with n decimal places.

```
round(,n)
```

Parameter	Description
	The value of this parameter is of the tinyint, smallint, integer, or bigint type.
n	The value of this parameter specifies the number of decimal places that you want the function to retain.

Same as the data type of the parameter value.

Compare the number of page views (PVs) of the current day with the number of PVs of the previous day. Then, present the comparison result as a percentage.

- Query statement

```
* | SELECT diff [1] AS today, round((diff [3] -1.0) * 100, 2) AS growth FROM (SELECT compare(pv, 8
6400) as diff FROM (SELECT COUNT(*) as pv FROM website_log))
```

- Query and analysis results

today	growth
1564075.0	-22.11

sign function

The sign function returns the sign of . Valid values: 1, 0, and -1.

```
sign()
```

Parameter	Description
	The value of this parameter is of the integer, smallint, tinyint, real, double, bigint, or decimal(p,s) type. • If is a positive number, the function returns 1. • If is 0, the function returns 0. • If is a negative number, the function returns -1.

Same as the data type of the parameter value.

Obtain the sign of 10.

- Query statement

```
* | SELECT sign(10)
```

- Query and analysis results

_col0
1

sin function

The sin function calculates the sine of .

```
sin()
```

Parameter	Description
	The value of this parameter is of the double type.

The double type.

Calculate the sine of the 90° angle.

- Query statement

```
* | select sin(pi()/2)
```

- Query and analysis results

sqrt function

The sqrt function calculates the square root of .

```
sqrt()
```

Parameter	Description
	The value of this parameter is of the double type.

The double type.

Calculate the square root of 100.

- Query statement

```
* | select sqrt(100)
```

- Query and analysis results

tan function

The tan function calculates the tangent of .

```
tan()
```

Parameter	Description
	The value of this parameter is of the double type.

The double type.

Calculate the tangent of the 30° angle.

- Query statement

```
* | SELECT tan(pi()/6)
```

- Query and analysis results

tanh function

The tanh function calculates the hyperbolic tangent of .

```
tanh()
```

Parameter	Description
	The value of this parameter is of the double type.

The double type.

Calculate the hyperbolic tangent of the 30° angle.

- Query statement

```
* | SELECT tanh(pi()/6)
```

- Query and analysis results

_col0
0.4804727781564516

to_base function

The to_base function converts to a base y string.

```
to_base(,)
```

Parameter	Description
	The value of this parameter is of the bigint type.
	The value of this parameter is of the bigint type. The value specifies a numeral system. Valid values: [2,36].

The varchar type.

Convert 180 to a binary string.

- Query statement

```
* | SELECT  to_base(180, 2)
```

- Query and analysis results

_col0
10110100

truncate function

The truncate function removes the fractional part of .

```
truncate()
```

Parameter	Description
	The value of this parameter is of the double type.

The double type.

Remove the fractional part of 11.11.

- Query statement

```
* | SELECT  truncate(11.11)
```

- Query and analysis results

width_bucket function

The width_bucket function returns the bucket number of .

- The following width_bucket function divides a numeric range into buckets of equal width and returns the bucket number of .

```
width_bucket(,bound1,bound2,numBuckets)
```

- The following width_bucket function returns the bucket number of in the range of buckets that are specified by an array.

```
width_bucket(,bins)
```

Parameter	Description
	The value of this parameter is of the double type.
bound1	The value of this parameter specifies the lower limit of the numeric range.
bound2	The value of this parameter specifies the upper limit of the numeric range.
numBuckets	The value of this parameter specifies the number of buckets. The value must be an integer greater than 0.
bins	The value of this parameter specifies the range of buckets. *The value is an array of the double type.*

The bigint type.

> **Note**
>
> - If is within the range, the function returns the bucket number of .
> - If is below the lower limit, the function returns 0.
> - If is above the upper limit, the function returns *numBuckets+1*.

- Example 1: Divide the range [10,80) into 7 buckets. Then, obtain the bucket number for each value of the request_time field.
  - Query statement

```
* | SELECT request_time, width_bucket(request_time, 10, 80,7) AS numBuckets
```

- Query and analysis results

request_time	numBuckets
26.0	2
49.0	4
10.0	1
34.0	3
40.0	4
77.0	7
28.0	2
66.0	6
23.0	2

- Example 2: Use an array to specify the range of 7 buckets. Then, obtain the bucket number for each value of the request_time field.
  - Query statement

```
* | SELECT request_time, width_bucket(request_time, array[10,20,30,40,50,60,70,80]) AS numBucket
s
```

  - Query and analysis results

request_time	numBuckets
26.0	2
13.0	1
28.0	2
30.0	3
67.0	6
12.0	1
54.0	5
10.0	1
45.0	4

8.1.11. Mathematical statistics functions

This topic describes the syntax of mathematical statistics functions. This topic also provides examples on how to use the functions.

Function	Syntax	Description
corr function	corr(,)	Returns the coefficient of correlation between and . The return value is in the range of [0,1].
covar_pop function	covar_pop(,)	Returns the population covariance of and .
covar_samp function	covar_samp(,)	Returns the sample covariance of and .

Function	Syntax	Description
regr_intercept function	regr_intercept(,)	Returns the y-intercept of the line for the linear equation that is determined by the (x,y) pair.
regr_slope function	regr_slope(,)	Returns the slope of the line for the linear equation that is determined by the (x,y) pair.
stddev function	stddev()	Returns the sample standard deviation of . This function is equivalent to the stddev_samp function.
stddev_samp function	stddev_samp()	Returns the sample standard deviation of .
stddev_pop function	stddev_pop()	Returns the population standard deviation of .
variance function	variance()	Returns the sample variance of . This function is equivalent to the var_samp function.
var_samp function	var_samp()	Returns the sample variance of .
var_pop function	var_pop()	Returns the population variance of .

corr function

The corr function returns the coefficient of correlation between and . A larger return value indicates a higher correlation.

```
corr(,)
```

Parameter	Description
	The value of this parameter is of the double type.
	The value of this parameter is of the double type.

The double type. The return value is in the range of [0,1].

Calculate the coefficient of correlation between the values of the request_length and request_time fields.

- Query statement

```
* | SELECT corr(request_length,request_time)
```

- Query and analysis results

covar_pop function

The covar_pop function returns the population covariance of and .

```
covar_pop(,)
```

Parameter	Description
	The value of this parameter is of the double type.

Parameter	Description
	The value of this parameter is of the double type.

The double type.

Calculate the population covariance of pretax profits and pretax turnovers in each minute.

- Query statement

```
*|
SELECT
  covar_pop(PretaxGrossAmount, PretaxAmount) AS "Population covariance",
  time_series(__time__, '1m', '%H:%i:%s', '0') AS time
GROUP BY
  time
```

- Query and analysis results

Population covariance	time
7.351615326821829	17:08:00

covar_samp function

The covar_samp function returns the sample covariance of and .

```
covar_samp(,)
```

Parameter	Description
	The value of this parameter is of the double type.
	The value of this parameter is of the double type.

The double type.

Calculate the sample covariance of pretax profits and pretax turnovers in each minute.

- Query statement

```
*|
SELECT
  covar_samp(PretaxGrossAmount, PretaxAmount) AS "Sample covariance",
  time_series(__time__, '1m', '%H:%i:%s', '0') AS time
GROUP BY
  time
```

- Query and analysis results

Sample covariance	time
2.2910940581194376	15:50:00
4.721554417070316	15:49:00

regr_intercept function

The regr_intercept function returns the y-intercept of the line for the linear equation that is determined by the (x,y) pair. is the dependent value. is the independent value.

```
regr_intercept(,)
```

Parameter	Description
	The value of this parameter is of the double type.
	The value of this parameter is of the double type.

The double type.

Calculate the y-intercept of the line for the linear equation that is determined by the values of the request_time and request_length fields.

- Query statement

```
* | SELECT regr_intercept(request_length,request_time)
```

- Query and analysis results

regr_slope function

The regr_slope function returns the slope of the line for the linear equation that is determined by the (x,y) pair. is the dependent value. is the independent value.

```
regr_slope(,)
```

Parameter	Description
	The value of this parameter is of the double type.
	The value of this parameter is of the double type.

The double type.

Calculate the slope of the line for the linear equation that is determined by the values of the request_time and request_length fields.

- Query statement

```
* | SELECT regr_slope(request_length,request_time)
```

- Query and analysis results

stddev function

The stddev function returns the sample standard deviation of . This function is equivalent to the stddev_samp function.

```
stddev()
```

Parameter	Description
	The value of this parameter is of the double or bigint type.

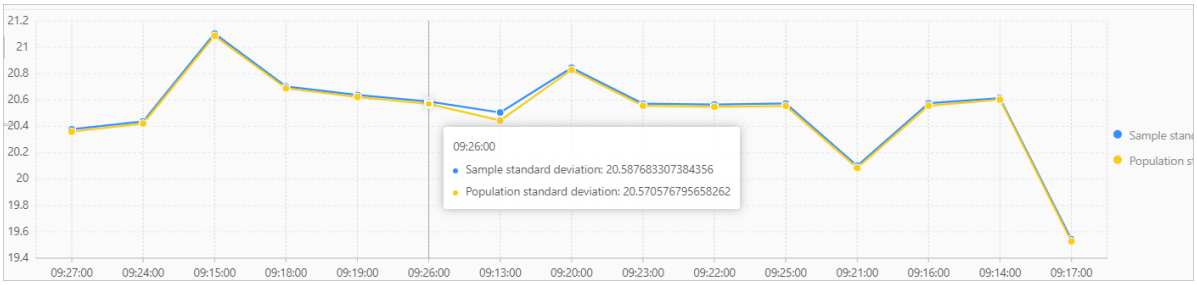
The double type.

Calculate the sample standard deviation and population standard deviation of pretax incomes and display the calculated values in a line chart.

- Query statement

```
* |
SELECT
  stddev(PretaxGrossAmount) as "Sample standard deviation",
  stddev_pop(PretaxGrossAmount) as "Population standard deviation",
  time_series(__time__, '1m', '%H:%i:%s', '0') AS time
GROUP BY
  time
```

- Query and analysis results



stddev_samp function

The stddev_samp function returns the sample standard deviation of .

```
stddev_samp()
```

Parameter	Description
	The value of this parameter is of the double or bigint type.

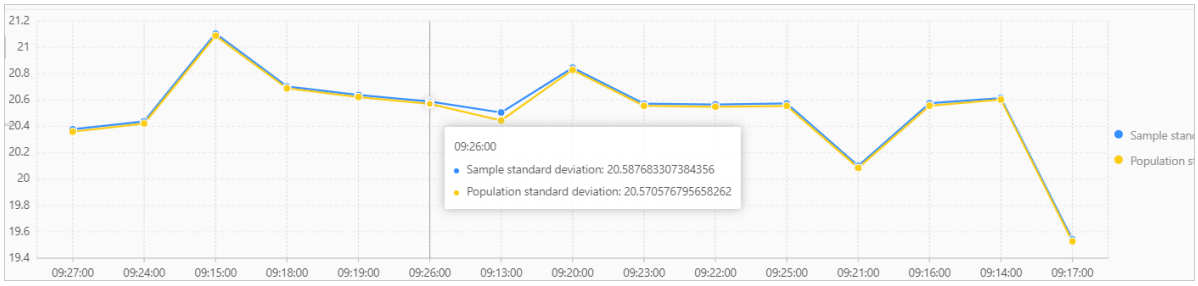
The double type.

Calculate the sample standard deviation and population standard deviation of pretax incomes and display the calculated values in a line chart.

- Query statement

```
* |
SELECT
  stddev_samp(PretaxGrossAmount) as "Sample standard deviation",
  stddev_pop(PretaxGrossAmount) as "Population standard deviation",
  time_series(__time__, '1m', '%H:%i:%s', '0') AS time
GROUP BY
  time
```

- Query and analysis results



stddev_pop function

The stddev_pop function returns the population standard deviation of .

```
stddev_pop()
```

Parameter	Description
	The value of this parameter is of the double or bigint type.

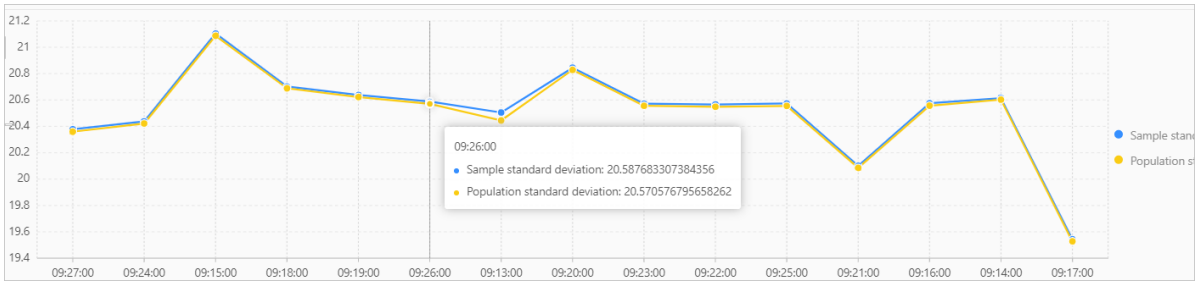
The double type.

Calculate the sample standard deviation and population standard deviation of pretax incomes and display the calculated values in a line chart.

- Query statement

```
* |
SELECT
  stddev(PretaxGrossAmount) as "Sample standard deviation",
  stddev_pop(PretaxGrossAmount) as "Population standard deviation",
  time_series(__time__, '1m', '%H:%i:%s', '0') AS time
GROUP BY
  time
```

- Query and analysis results



variance function

The variance function returns the sample variance of . This function is equivalent to the var_samp function.

```
variance()
```

Parameter	Description
	The value of this parameter is of the double or bigint type.

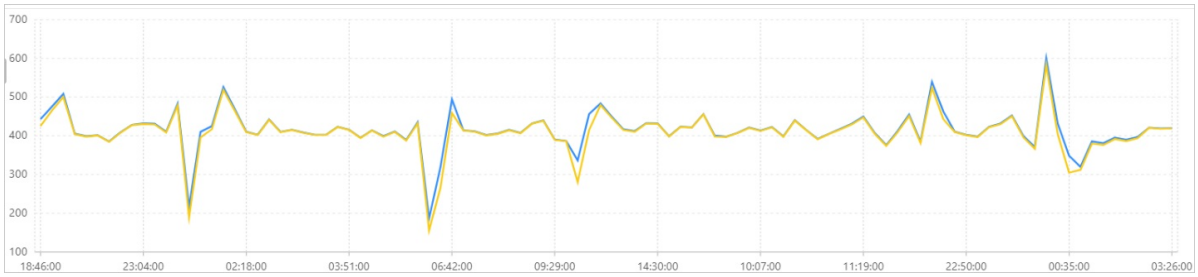
The double type.

Calculate the sample variance and population variance of pretax incomes and display the calculated values in a line chart.

- Query statement

```
* |
SELECT
  variance(PretaxGrossAmount) as "Sample variance",
  var_pop(PretaxGrossAmount) as "Population variance",
  time_series(__time__, '1m', '%H:%i:%s', '0') as time
GROUP BY
  time
```

- Query and analysis results



var_samp function

The var_samp function returns the sample variance of .

```
var_samp()
```

Parameter	Description
	The value of this parameter is of the double or bigint type.

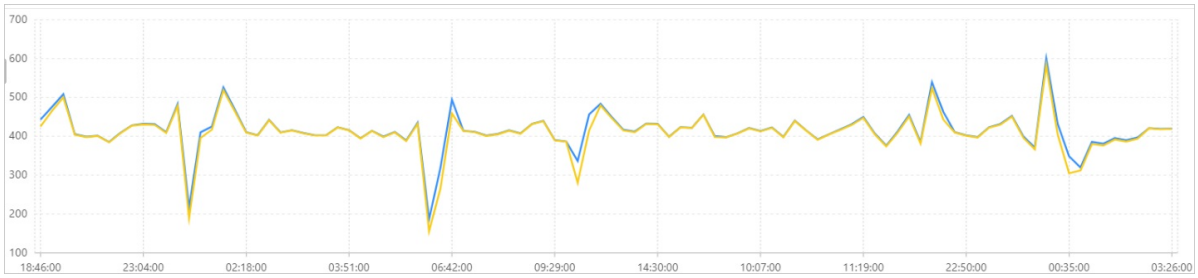
The double type.

Calculate the sample variance and population variance of pretax incomes and display the calculated values in a line chart.

- Query statement

```
* |
SELECT
  var_samp(PretaxGrossAmount) as "Sample variance",
  var_pop(PretaxGrossAmount) as "Population variance",
  time_series(__time__, '1m', '%H:%i:%s', '0') as time
GROUP BY
  time
```

- Query and analysis results



var_pop function

The var_pop function returns the population variance of .

```
var_pop()
```

Parameter	Description
	The value of this parameter is of the double or bigint type.

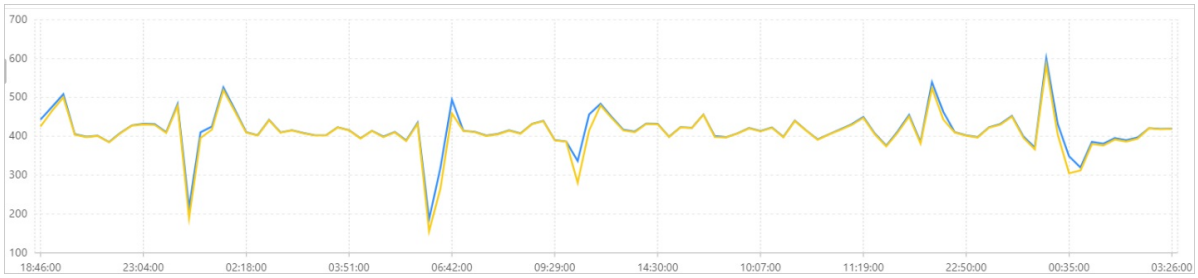
The double type.

Calculate the sample variance and population variance of pretax incomes and display the calculated values in a line chart.

- Query statement

```
* |
SELECT
  variance(PretaxGrossAmount) as "Sample variance",
  var_pop(PretaxGrossAmount) as "Population variance",
  time_series(__time__, '1m', '%H:%i:%s', '0') as time
GROUP BY
  time
```

- Query and analysis results



8.1.12. Data type conversion functions

You can use type conversion functions to convert data to the correct type in a query statement.

Function	Syntax	Description
cast function	cast(as *type*)	Converts the values of the *x* field as a data type. If the cast function cannot convert a value, the query that calls this function fails.

Function	Syntax	Description
try_cast function	try_cast(as *type*)	Converts the values of the *x* as a data type. If the try_cast function cannot convert a value, the function returns NULL. The query that calls this function can process the NULL value and continue to run. **Note** A log may contain data of the data types that you do not expect. When you query logs, we recommend that you use the try_cast function. This way, conversion failures will not cause your queries fail.
typeof function	typeof()	Returns the data type of the *x* field.

cast function

The cast function is used to convert the values of the *x* field to a specified data type. If the cast function cannot convert a value, the query that calls this function fails.

```
cast(x as type)
```

Parameter	Description
x	The value of this parameter can be of any data type.
type	A SQL data type. Valid values: bigint, varchar, double, boolean, timestamp, decimal, array, and map. Example: `cast(json_parse( key) as array(varchar))` . Each SQL data type maps to a data type that you can use in a Log Service index. For information about the mapping, see Data type mappings.

The data type that is specified by the *type* parameter.

Converts number 1 to the boolean data type.

- Query statement

```
* | select cast(1 as boolean)
```

- Query result

try_cast function

Converts the values of the *x* field to a specified data type. If the try_cast function cannot convert a value, the function returns NULL. The query that calls this function can process the NULL value and continue to run.

```
try_cast(x as type)
```

Parameter	Description
x	The value of this parameter can be of any data type.
type	A SQL data type. Valid values: bigint, varchar, double, boolean, timestamp, decimal, array, and map. Example: `try_cast(json_parse( key) as map(varchar, varchar))` . Each SQL data type maps to a data type that you can use in a Log Service index. For information about the mapping, see Data type mappings.

The data type that the *type* parameter specifies.

Converts the values of the uid field to the varchar data type.

- Query statement

```
* | select try_cast(uid as varchar)
```

- Query result

typeof function

Returns the data type of the *x* field.

```
typeof(x)
```

Parameter	Description
x	The value of this parameter can be of any data type.

Data of the varchar data type.

Returns the data type of the request_time field.

- Query statement

```
* |SELECT typeof(request_time)
```

- Query result

Data type mappings

The following table describes the mappings between the SQL data types and the data types that are supported by a Log Service index.

Data types that are supported by Log Service index	SQL data type
long	bigint
text	varchar
double	double
json	varchar

8.1.13. Security check functions

Log Service provides security check functions based on the globally shared asset library of WhiteHat Security. You can use security check functions to check whether an IP address, a domain name, or a URL in a log is secure. This topic describes the syntax of security check functions. This topic also provides examples on how to use security check functions.

Scenarios

You can use security check functions in the following scenarios:

- Enterprises and institutions in industries, such as Internet, gaming, and consulting, require robust O&M services. The enterprises and institutions can use security check functions to identify suspicious requests or attacks, perform in-depth analysis, and defend against potential attacks.
- Enterprises and institutions in industries, such as banking, securities, and e-commerce, require strong protection for internal assets. The enterprises and institutions can use security check functions to identify access to suspicious websites and identify download of trojans. This way, the enterprises and institutions can prevent security risks at the earliest opportunity.

Features

Security check functions provide the following features:

- Reliability: Security check functions are based on the globally shared asset library of WhiteHat Security. When WhiteHat Security is updated, the security check functions are also updated.
- Efficiency: Security check functions can check millions of IP addresses, domain names, and URLs within seconds.
- Ease of use: You can use the security_check_ip, security_check_domain, and security_check_url functions to analyze network logs.
- Flexibility: You can perform interactive queries, visualize query and analysis results, and configure alerts.

Functions

The following table describes the security check functions that are supported by Log Service.

Notice

Function	Syntax	Description
security_check_ip function	security_check_ip()	Checks whether an IP address is secure.
security_check_domain function	security_check_domain()	Checks whether a domain name is secure.
security_check_url function	security_check_url()	Checks whether a URL is secure.

security_check_ip function

The security_check_ip function is used to check whether an IP address is secure.

```
security_check_ip()
```

Parameter	Description
	The value of this parameter is an IP address.

The bigint type. Valid values:

- 1: The specified IP address is suspicious.
- 0: The specified IP address is secure.

Query suspicious clients that access a website based on the client_ip field.

- Query statement

```
* |
SELECT
  client_ip,
  ip_to_country(client_ip,'en') AS country,
  ip_to_provider(client_ip) AS provider,
  count(1) AS PV
WHERE
  security_check_ip(client_ip) = 1
GROUP BY
  client_ip
ORDER BY
  PV DESC
```

- Query and analysis result

client_ip	country	provider	PV
180 3	CN		3
103	CN		3
180 7	CN		1

security_check_domain function

The security_check_domain function is used to check whether a domain name is secure.

```
security_check_domain()
```

Parameter	Description
	The value of this parameter is a domain name.

The bigint type. Valid values:

- 1: The specified domain name is suspicious.
- 0: The specified domain name is secure.

Calculate the number of times that a website is accessed by suspicious domain names per minute. The query and analysis result is displayed in a line chart.

- Query statement

```
status : * |
SELECT
  count_if(
    security_check_domain (http_referer) != 0
  ) AS "Total Issues",
  time_series(__time__, '1m', '%H:%i:%s', '0') AS time
GROUP BY
  time
```

- Query and analysis result

security_check_url function

The security_check_url function is used to check whether a URL is secure.

```
security_check_url()
```

Parameter	Description
	The value of this parameter is a URL.

The bigint type. Valid values:

- 1: The specified URL is suspicious.
- 0: The specified URL is secure.

Calculate the number of times that a website is accessed by secure URLs per minute. The query and analysis result is displayed in a line chart.

- Query statement

```
status : * |
SELECT
  count_if(
    security_check_url (request_uri) = 0
  ) AS "Total Issues",
  time_series(__time__, '1m', '%H:%i', '0') as time
GROUP BY
  time
LIMIT
  20
```

- Query and analysis result

8.1.14. Window functions

This topic describes the syntax of window functions and provides examples on how to use window functions.

Summary

Aggregate functions calculate the single result for a group of rows, and window functions calculate the result for each row in a group. A window function has three elements: partition, order, and frame. For more information, see Window Function Concepts and Syntax.

```
function over (
    [partition by partition_expression]
    [order by order_expression]
    [frame]
)
```

- Partition: The partition element is defined by the PARTITION BY clause. The PARTITION BY clause separates rows into partitions. If you do not specify the PARTITION BY clause, all rows are treated as a single partition.
- Order: The order element is defined by the ORDER BY clause. The ORDER BY clause sorts rows in all partitions.

  > **Note** If you use the ORDER BY clause to sort rows on fields that have the same value, the order of these rows is non-deterministic. You can include additional fields in the ORDER BY clause to obtain the expected order of these rows. Example: `order by request_time, request_method`.

- Frame: The frame element is defined by the FRAME clause. The FRAME clause specifies a subset of each partition. A frame further refines the rows in each partition. You cannot specify the FRAME clause for ranking functions. Syntax of the FRAME clause: `{ rows | range} { frame_start | frame_between }`. Example: `rang`

`e between unbounded preceding and unbounded following` . For more information, see Window Function Frame Specification.

Window functions

Category	Function	Syntax	Description
Aggregate functions	Aggregate functions	None	You can use all aggregate functions as window functions. For more information about aggregate functions, see Aggregate function.
Ranking functions	cume_dist function	cume_dist()	Calculate the cumulative distribution of each value in a partition. The result is obtained by using division. The numerator is the number of rows whose field values are smaller or equal to the field value of the specified row. The specified row is also counted. The denominator is the total number of rows in the partition. The calculation is based on the order of the rows in the partition. Value range: (0,1].
	dense_rank function	dense_rank()	Calculates the rank of each value in a partition. Rows that have the same field value are assigned the same rank. The ranks are consecutive. For example, if two rows have the same rank of 1, the rank of the next row is 2.
	ntile function	ntile(*n*)	Divide the rows in each partition into the number of groups specified by the N parameter.
	percent_rank function	percent_rank()	Calculates the percentage ranking of each row in a partition.
	rank function	rank()	Calculate the rank of each row in a partition. Rows that have the same field value are assigned the same rank. The ranks are not consecutive. For example, if two rows have the same rank of 1, the rank of the next row is 3.
	row_number function	row_number()	Calculate the rank of each row in a partition. The ranks are unique, and start from 1. Rows that have the same value are assigned consecutive ranks. For example, three rows with the same field value are assigned the ranks 1, 2, and 3.
	first_value function	first_value()	Returns the value of the specified field in the first row of each partition.
	last_value function	last_value()	Returns the value of the specified field in the last row of each partition.

Category	Function	Syntax	Description
Offset functions	lag function	lag(,*offset*,*defaut_value*)	Returns the value of the specified field in the row that is at the specified offset before the current row of each partition. The offset is specified by the *offset* parameter. If no row exists at the specified offset before the current row, the value that is specified by the *defaut_value* parameter is returned for the partition.
	lead function	lead(,*offset*,*defaut_value*)	Returns the value of the specified field in the row that is at the specified offset after the current row of each partition. The offset is specified by the *offset* parameter. If no row exists at the specified offset after the current row, the value that is specified by the *defaut_value* is returned for the partition.
	nth_value function	nth_value(, *offset*)	Returns the value of the specified field in the row that is at the specified offset from the beginning of each partition. The offset is specified by the *offset* parameter.

Aggregate functions

You can use all aggregate functions as window functions. For more information about aggregate functions, see Aggregate function. The following example shows how to use the sum function as a window function.

```
sum() over (
    [partition by partition_expression]
    [order by order_expression]
    [frame]
)
```

Parameter	Description
partition by *partition_expression*	Specifies how the rows are partitioned based on the value of the partition_expression parameter.
order by *order_expression*	Specifies how the rows in each partition are ordered based on the value of the order_expression parameter.
frame	Specifies a subset of each partition. Example: `range between unbounded preceding and unbounded following`.

The double data type.

Calculate the percentage of each employee salary in each department.

- Query statement

```
* |
SELECT
  department,
  staff_name,
  salary,
  round ( salary * 1.0 / sum(salary) over(partition by department), 3) AS salary_percentage
```

- Query result

department	staff_name	salary	salary_percentage
dev	Rob	9000	0.277
dev	Blan	8500	0.262
dev	Sansa	8000	0.246
dev	Snow	7000	0.215
Marketing	Achilles	8500	0.362
Marketing	San	8000	0.340
Marketing	Blan	7000	0.298

cume_dist function

Calculates the cumulative distribution of each value in a partition. The result is obtained by using division. The numerator is the number of rows whose field values are smaller or equal to the field value of the specified row. The specified row is also counted. The denominator is the total number of rows in the partition. The calculation is based on the order of the rows in the partition. Value range: (0,1].

```
cume_dist() over (
    [partition by partition_expression]
    [order by order_expression]
)
```

Parameter	Description
partition by *partition_expression*	Specifies how the rows are partitioned based on the value of the partition_expression parameter.
order by *order_expression*	Specifies how the rows in each partition are ordered. The rows are ordered based on the value of the order_expression parameter.

The double data type.

Calculate the cumulative distribution of the size of each object in an OSS bucket named bucket00788.

- Query statement

```
bucket=bucket00788 |
select
  object,
  object_size,
  cume_dist() over (
    partition by object
    order by
      object_size
  ) as cume_dist
from  oss-log-store
```

- Query result

object	object_size	cume_dist
dashboard%2F2020%2F05%2F20%2F16%2F47.csv	4591	0.5
dashboard%2F2020%2F05%2F20%2F16%2F47.csv	6469	1.0
oss-log-562[redacted]11-cn-hangzhou%2Foss-log-store_oss_access_center_en%2Freport-1581933277-637788	1526	0.3333333333333333
oss-log-56[redacted]911-cn-hangzhou%2Foss-log-store_oss_access_center_en%2Freport-1581933277-637788	1921	0.6666666666666666
oss-log-56[redacted]11-cn-hangzhou%2Foss-log-store_oss_access_center_en%2Freport-1581933277-637788	6074	1.0
245-da918c.model	1818	0.08333333333333333
245-da918c.model	1854	0.16666666666666667

dense_rank function

Calculates the rank of each value in a partition. Rows that have the same field value are assigned the same rank. The ranks are consecutive. For example, if two rows have the same rank of 1, the rank of the next row is 2.

```
dense_rank() over (
    [partition by partition_expression]
    [order by order_expression]
)
```

Parameter	Description
partition by *partition_expression*	Specifies how the rows are partitioned based on the value of the partition_expression parameter.
order by *order_expression*	Specifies how the rows in each partition are ordered based on the value of the order_expression parameter.

The bigint data type.

Calculate the rank of each employee salary in each department.

- Query statement

```
* |
select
  department,
  staff_name,
  salary,
  dense_rank() over(
    partition by department
    order by
      salary desc
  ) as salary_rank
order by
  department,
  salary_rank
```

- Query result

department	staff_name	salary	salary_rank
Marketing	Blan Stark	9000	1
Marketing	Smith	9000	1
Marketing	Achilles	8000	2
dev	Rob	9000	1
dev	Blan	8500	2
dev	Sansa	8000	3

ntile function

Divides the rows in each partition into a number of groups. The number of groups is specified by the N parameter.

```
ntile(n) over (
    [partition by partition_expression]
    [order by order_expression]
)
```

Parameter	Description
n	Specifies the number of groups.
partition by *partition_expression*	Specifies how the rows are partitioned based on the value of the partition_expression parameter.
order by *order_expression*	Specifies how the rows in each partition are ordered based on the value of the order_expression parameter.

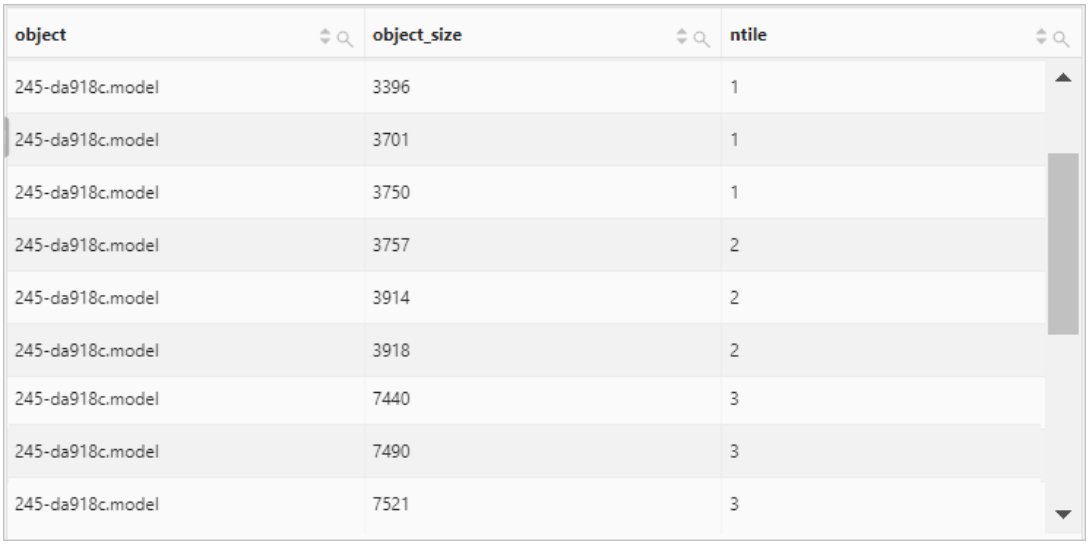
The bigint data type.

Divide the rows in each partition into three groups.

- Query statement

```
object=245-da918c.model |
select
  object,
  object_size,
  ntile(3) over (
    partition by object
    order by
      object_size
  ) as ntile
from  oss-log-store
```

- Query result



object	object_size	ntile
245-da918c.model	3396	1
245-da918c.model	3701	1
245-da918c.model	3750	1
245-da918c.model	3757	2
245-da918c.model	3914	2
245-da918c.model	3918	2
245-da918c.model	7440	3
245-da918c.model	7490	3
245-da918c.model	7521	3

percent_rank function

Calculates the percentage ranking of each row in a partition. The calculation formula is `(rank - 1)/(total_rows - 1)`. In the formula, rank represents the rank of the current row, and total_rows represents the total number of rows in a partition.

```
percent_rank() over (
    [partition by partition_expression]
    [order by order_expression]
)
```

Parameter	Description
partition by *partition_expression*	Specifies how the rows are partitioned based on the value of the partition_expression parameter.
order by *order_expression*	Specifies how the rows in each partition are ordered based on the value of the order_expression parameter.

The double data type.

Calculate the percentage ranking of the size of each OSS object.

- Query statement

```
object=245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model|
select
  object,
  object_size,
 percent_rank() over (
    partition by object
    order by
      object_size
  ) as ntile
FROM  oss-log-store
```

- Query result

object	object_size	ntile
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	7442	0.0
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	7635	0.2
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	8221	0.4
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	8272	0.6
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	8706	0.8
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	8988	1.0

rank function

Calculates the rank of each row in a partition. Rows that have the same field value are assigned the same rank. The ranks are not consecutive. For example, if two rows have the same rank of 1, the rank of the next row is 3.

```
rank() over (
    [partition by partition_expression]
    [order by order_expression]
)
```

Parameter	Description
partition by *partition_expression*	Specifies how the rows are partitioned based on the value of the partition_expression parameter.
order by *order_expression*	Specifies how the rows in each partition are ordered based on the value of the order_expression parameter.

The bigint data type.

Calculate the rank of each employee salary in each department.

- Query statement

```
* |
select
  department,
  staff_name,
  salary,
  rank() over(
    partition by department
    order by
      salary desc
  ) as salary_rank
order by
  department,
  salary_rank
```

- Query result

department	staff_name	salary	salary_rank
Marketing	Blan Stark	9000	1
Marketing	Smith	9000	1
Marketing	Achilles	8000	3
dev	Rob	9000	1
dev	Blan	8500	2
dev	Sansa	8000	3

row_number function

Calculate the rank of each row in a partition. The ranks are unique, and start from 1.

```
row_number() over (
    [partition by partition_expression]
    [order by order_expression]
)
```

Parameter	Description
partition by *partition_expression*	Specifies how the rows are partitioned based on the value of the partition_expression parameter.
order by *order_expression*	Specifies how the rows in each partition are ordered. The rows are ordered based on the value of the order_expression parameter.

The bigint data type.

Calculate the rank of each employee on the salary in each department.

- Query statement

```
* |
select
  department,
  staff_name,
  salary,
  row_number() over(
    partition by department
    order by
      salary desc
  ) as salary_rank
order by
  department,
  salary_rank
```

- Query result

department	staff_name	salary	salary_rank
Marketing	Blan Stark	9000	1
Marketing	Smith	9000	2
Marketing	Achilles	8000	3
dev	Rob	9000	1
dev	Blan	8500	2
dev	Sansa	8000	3

first_value function

Returns the value of the specified field in the first row of each partition.

```
first_value() over (
    [partition by partition_expression]
    [order by order_expression]
    [frame]
)
```

Parameter	Description
	The field name. The field can be of any data type.
partition by *partition_expression*	Specifies how the rows are partitioned based on the value of the partition_expression parameter.
order by *order_expression*	Specifies how the rows in each partition are ordered based on the value of the order_expression parameter.
frame	Specifies a subset of each partition. Example: `range between unbounded preceding and unbounded following`.

The data type is the same as the data type of the parameter.

Return the minimum size of each object in the specified OSS bucket.

- Query statement

```
bucket :bucket90 |
select
  object,
  object_size,
  last_value(object_size) over (
    partition by object
    order by
      object_size
     range between unbounded preceding and unbounded following
  ) as last_value
from  oss-log-store
```

- Query result

object	object_size	first_value
oss-log-56[redacted]11-cn-hangzhou%2Foss-log-store_oss_access_center_en%2Freport-1581933277-637788.	1157	1157
oss-log-562[redacted]11-cn-hangzhou%2Foss-log-store_oss_access_center_en%2Freport-1581933277-637788..	6751	1157
dashboard%2F2020%2F05%2F20%2F16%2F47.csv	6949	6949
dashboard%2F2020%2F05%2F20%2F16%2F47.csv	8749	6949
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	1195	1195
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	5775	1195
245-da918c3e2dd9dc9cb4d9283b%2F555e2441b6a4c7f094099a6dba8e7a5f.model	5856	1195

last_value function

Returns the value of the specified field in the last row of each partition.

```
last_value() over (
    [partition by partition_expression]
    [order by order_expression]
    [frame]
)
```

Parameter	Description
	The field name. The field can be of any data type.
partition by *partition_expression*	Specifies how the rows are partitioned based on the value of the partition_expression parameter.
order by *order_expression*	Specifies how the rows in each partition are ordered based on the value of the order_expression parameter.
frame	Specifies a subset of each partition. Example: `range between unbounded preceding and unbounded following`.

The data type is the same as the data type of the parameter.

Return the maximum size of each object in the specified OSS bucket.

- Query statement

```
bucket :bucket90 |
select
  object,
  object_size,
  last_value(object_size) over (
    partition by object
    order by
      object_size
     range between unbounded preceding and unbounded following
  ) as last_value
from  oss-log-store
```

- Query result

object	object_size	last_value
245-da918c.model	2383	6936
245-da918c.model	2975	6936
245-da918c.model	3375	6936
245-da918c.model	4999	6936
245-da918c.model	5199	6936
245-da918c.model	6125	6936
245-da918c.model	6936	6936
dashboard%2F2020%2F05%2F20%2F16%2F47.csv	2435	2603
dashboard%2F2020%2F05%2F20%2F16%2F47.csv	2603	2603

lag function

Returns the value of the specified field in the row that is at the specified offset before the current row in each partition. The offset is specified by the *offset* parameter.

```
lag(,offset,defaut_value) over (
    [partition by partition_expression]
    [order by order_expression]
    [frame]
)
```

Parameter	Description
	The field name. The field can be of any data type.
offset	The offset before the current row in a partition. If the value of the *offset* parameter is 0, the value of the specified field in the current row is returned.
defaut_value	If no row exists at the specified offset before the current row, the value of the *defaut_value* parameter is returned.
partition by *partition_expression*	Specifies how the rows are partitioned based on the value of the partition_expression parameter.
order by *order_expression*	Specifies how the rows in each partition are ordered based on the value of the order_expression parameter.
frame	Specifies a subset of each partition. Example: `range between unbounded preceding and unbounded following`.

The data type is the same as the data type of the parameter.

Count the daily unique visitors (UVs) to your website and calculates the percentage of the increase in UVs over the previous day.

- Query statement

```
* |
select
  day,
  UV,
  UV * 1.0 /(lag(UV, 1, 0) over()) as diff_percentage
from  (
    select
      approx_distinct(client_ip) as UV,
      date_trunc('day', __time__) as day
    from  log
    group by
      day
    order by
      day asc
  )
```

- Query result

day	UV	diff_percentage
2021-08-02 00:00:00.000	184332	Infinity
2021-08-03 00:00:00.000	386788	2.098322591845149
2021-08-04 00:00:00.000	377834	0.9768503676432567
2021-08-05 00:00:00.000	366409	0.9697618530889226
2021-08-06 00:00:00.000	390285	1.0651621548597333
2021-08-07 00:00:00.000	373103	0.9559757613026378
2021-08-08 00:00:00.000	386960	1.037139878264179
2021-08-09 00:00:00.000	226578	0.5855333884639239

lead function

Returns the value of the specified field in the row that is at the specified offset after the current row in each partition. The offset is specified by the *offset* parameter.

```
lead(,offset,defaut_value) over (
    [partition by partition_expression]
    [order by order_expression]
    [frame]
)
```

Parameter	Description
	The field name. The field can be of any data type.
offset	The offset after the current row in a partition. If the value of the *offset* parameter is 0, the value of the specified field in the current row is returned.
defaut_value	If no row exists at the specified offset after the current row, the value of the *defaut_value* parameter is returned.
partition by *partition_expression*	Specifies how the rows are partitioned based on the value of the partition_expression parameter.
order by *order_expression*	Specifies how the rows in each partition are ordered based on the value of the order_expression parameter.
frame	Specifies a subset of each partition. Example: `range between unbounded preceding and unbounded following`.

The data type is the same as the data type of the parameter.

Count the hourly unique visitors (UVs) to your website on 2021-08-26 and calculates the difference in percentage between UVs of two consecutive hours.

- Query statement

```
* |
select
  time,
  UV,
  UV * 1.0 /(lead(UV, 1, 0) over()) as diff_percentage
from  (
    select
      approx_distinct(client_ip) as uv,
      date_trunc('hour', __time__) as time
    from    log
    group by
      time
    order by
      time asc
  )
```

- Query result

time	UV	diff_percentage
2021-08-26 00:00:00.000	1185	0.11647336347552585
2021-08-26 01:00:00.000	10174	0.37027331950358485
2021-08-26 02:00:00.000	27477	0.8010787172011662
2021-08-26 03:00:00.000	34300	1.3467352467705838
2021-08-26 04:00:00.000	25469	2.646404821280133
2021-08-26 05:00:00.000	9624	7.947151114781173
2021-08-26 06:00:00.000	1211	0.11748156771439658
2021-08-26 07:00:00.000	10308	0.37978041411834059

nth_value function

Returns the value of the specified field in the row that is at the specified offset from the beginning of each partition. The offset is specified by the *offset* parameter.

```
nth_value(,offset) over (
    [partition by partition_expression]
    [order by order_expression]
    [frame]
)
```

Parameter	Description
	The field name. The field can be of any data type.
offset	The offset from the beginning of a partition.
partition by *partition_expression*	Specifies how the rows are partitioned based on the value of the partition_expression parameter.
order by *order_expression*	Specifies how the rows in each partition are ordered based on the value of the order_expression parameter.

Parameter	Description
frame	Specifies a subset of each partition. Example: `range between unbounded preceding and unbounded following`.

The data type is the same as the data type of the parameter.

Return the employees whose salary is the second highest in each department.

- Query statement

```
* |
select
  department,
  staff_name,
  salary,
  nth_value(staff_name, 2) over(
    partition by department
    order by
      sallary desc
      range between unbounded preceding and unbounded following
  ) as second_highest_sallary from log
```

- Query result

department	staff_name	salary	second_highest_salary
dev	Rob	9000	Blan '
dev	Blan	8500	Blan
dev	Sansa	8000	Blan
dev	Snow	7000	Blan
Marketing	Achilles	8500	San
Marketing	San	8000	San
Marketing	Blan	7000	San

8.1.15. IP functions

This topic describes the syntax of IP functions and provides examples on how to use the functions.

The following table describes the IP functions that are supported by Log Service.

Notice

Type	Function	Syntax	Description
	ip_to_city function	ip_to_city()	Identifies the city to which an IP address belongs. The returned result is the Chinese name of a city.
		ip_to_city(,'en')	Identifies the city to which an IP address belongs. The returned result is the administrative region code of a city.

Type	Function	Syntax	Description
IP functions	ip_to_city_geo function	ip_to_city_geo()	Identifies the longitude and latitude of the city to which an IP address belongs. This function returns the longitude and latitude of a city. Each city has only one set of coordinates.
	ip_to_country function	ip_to_country()	Identifies the country or the region to which an IP address belongs. The returned result is the Chinese name of a country or a region.
		ip_to_country(,'en')	Identifies the country or the region to which an IP address belongs. The returned result is the code of a country or a region.
	ip_to_country_code function	ip_to_country_code()	Identifies the country or the region to which an IP address belongs. The returned result is the code of a country or a region.
	ip_to_domain function	ip_to_domain()	Checks whether an IP address is an internal IP address or an external IP address.
	ip_to_geo function	ip_to_geo()	Identifies the longitude and latitude of the location where an IP address belongs.
	ip_to_provider function	ip_to_provider()	Identifies the Internet service provider (ISP) of an IP address.
	ip_to_province function	ip_to_province()	Identifies the state to which an IP address belongs. The returned result is the Chinese name of a state.
		ip_to_province(,'en')	Identifies the state to which an IP address belongs. The returned result is the administrative region code of a state.
	ip_prefix function	ip_prefix(,*prefix_bits*)	Obtains the prefix of an IP address.
	is_prefix_subnet_of function	is_prefix_subnet_of(,)	Checks whether a CIDR block is a subnet of a specified CIDR block.

Type	Function	Syntax	Description
CIDR block functions	is_subnet_of function	is_subnet_of(,)	Checks whether an IP address is in a specified CIDR block.
	ip_subnet_max function	ip_subnet_max()	Obtains the largest IP address in a CIDR block.
	ip_subnet_min function	ip_subnet_min()	Obtains the smallest IP address in a CIDR block.
	ip_subnet_range function	ip_subnet_range()	Obtains a CIDR block.

ip_to_city function

The ip_to_city function is used to identify the city to which an IP address belongs.

- The returned result is the Chinese name of a city.

```
ip_to_city()
```

- The returned result is the administrative region code of a city.

```
ip_to_city(,'en')
```

Parameter	Description
	The value is an IP address.

The varchar type.

Calculate the average processing time of requests, maximum processing time of requests, and IDs of requests that require the maximum processing time by city.

- Query statement

```
* |
SELECT
  AVG(request_time) AS avg_request_time,
  MAX(request_time) AS max_request_time,
  MAX_BY(requestId, request_time) AS requestId,
  ip_to_city(client_ip) AS city
GROUP BY
  city
```

- Query and analysis result

avg_request_time	max_request_time	requestId	city
44.84665110432753	80.0	i-02	
46.47286821705426	80.0	i-01	
43.87214611872146	80.0	i-01	

ip_to_city_geo function

The ip_to_city_geo function is used to identify the longitude and latitude of the city to which an IP address belongs. This function returns the longitude and latitude of a city. Each city has only one set of coordinates.

```
ip_to_city_geo()
```

Parameter	Description
	The value is an IP address.

The varchar type. Format: `latitude,longitude`.

Query the longitude and latitude of an IP address and view the distribution of clients.

- Query statement

```
* |
SELECT
  count(*) AS PV,
  ip_to_city_geo(client_ip) AS geo
GROUP BY
  geo
ORDER BY
  PV DESC
```

- Query and analysis result

PV	geo
9113	39.9288,116.389
5784	31.2222,121.458060

ip_to_country function

The ip_to_country function is used to identify the country or the region to which an IP address belongs.

- The returned result is the Chinese name of a country or a region.

```
ip_to_country()
```

- The returned result is the code of a country or a region.

```
ip_to_country(,'en')
```

Parameter	Description
	The value is an IP address.

The varchar type.

Calculate the average processing time of requests, maximum processing time of requests, and IDs of requests that require the maximum processing time by country or region.

- Query statement

```
* |
SELECT
  AVG(request_time) AS avg_request_time,
  MAX(request_time) AS max_request_time,
  MAX_BY(requestId, request_time) AS requestId,
  ip_to_country(client_ip) AS country
GROUP BY
  country
```

- Query and analysis result

avg_request_time	max_request_time	requestId	country
45.26589740373761	80.0	i-02	
47.07692307692308	80.0	i-01	
39.392857142857149	77.0	i-01	

ip_to_country_code function

The ip_to_country_code function is used to identify the country or the region to which an IP address belongs. The returned result is the code of a country or a region.

```
ip_to_country_code()
```

Parameter	Description
	The value is an IP address.

The varchar type.

Calculate the average processing time of requests, maximum processing time of requests, and IDs of requests that require the maximum processing time by country or region.

- Query statement

```
* |
SELECT
  AVG(request_time) AS avg_request_time,
  MAX(request_time) AS max_request_time,
  MAX_BY(requestId, request_time) AS requestId,
  ip_to_country_code(client_ip) AS country
GROUP BY
  country
```

- Query and analysis result

avg_request_time	max_request_time	requestId	country
45.01220462217606	80.0	i-01	MO
46.285714285714288	77.0	i-02	GB

ip_to_domain function

The ip_to_domain function is used to check whether an IP address is an internal IP address or an external IP address.

```
ip_to_domain()
```

Parameter	Description
	The value is an IP address.

The varchar type. Valid values: intranet and internet.

- intranet: an internal IP address.
- internet: an external IP address.

Calculate the total number of requests that are not from the internal network.

- Query statement

```
* | SELECT count(*) AS PV where ip_to_domain(client_ip)!='intranet'
```

- Query and analysis result

PV
941786

ip_to_geo function

The ip_to_geo function is used to identify the longitude and latitude of an IP address. For information about geohash functions, see Geo functions.

```
ip_to_geo()
```

Parameter	Description
	The value is an IP address.

The varchar type. Format: `latitude,longitude`.

Query the longitude and latitude of an IP address and view the distribution of clients.

- Query statement

```
* |
SELECT
  count(*) AS PV,
  ip_to_geo(client_ip) AS geo
GROUP BY
  geo
ORDER BY
  PV DESC
```

- Query and analysis result

PV	geo
5122	39.1423,117.173
4960	29.5569,106.553

ip_to_provider function

The ip_to_provider function is used to identify the ISP of an IP address.

```
ip_to_provider()
```

Parameter	Description
	The value is an IP address.

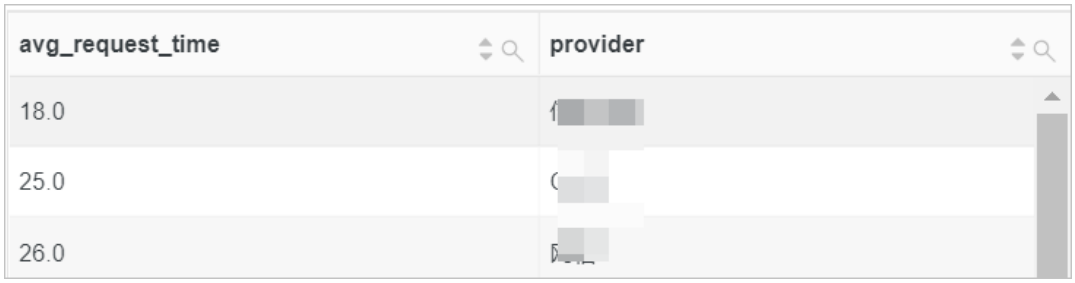
The varchar type.

Calculate the average processing time of requests from different ISPs.

- Query statement

```
* |
SELECT
  avg(request_time) AS avg_request_time,
  ip_to_provider(client_ip) AS provider
GROUP BY
  provider
ORDER BY
  avg_request_time
```

- Query and analysis result



ip_to_province function

The ip_to_province is used to identify the state to which an IP address belongs.

- The returned result is the Chinese name of a state.

```
ip_to_province()
```

- The returned result is the administrative region code of a state.

```
ip_to_province(,'en')
```

Parameter	Description
	The value is an IP address.

The varchar type.

Query the top 10 states from which the most requests are sent.

- Query statement

```
* | SELECT count(*) as PV, ip_to_province(client_ip) AS province GROUP BY province ORDER BY PV des
c LIMIT 10
```

If you want to exclude the requests that are sent from the internal network when you query the top 10 states, execute the following query statement:

```
* | SELECT count(*) AS PV, ip_to_province(client_ip) AS province WHERE ip_to_domain(client_ip) !=
'intranet'  GROUP BY province ORDER BY PV DESC LIMIT 10
```

- Query and analysis result

ip_prefix function

The ip_prefix function is used to obtain the prefix of an IP address. The returned result is an IP address in the subnet mask format, for example, 192.168.1.0/24.

```
ip_prefix(,prefix_bits)
```

Parameter	Description
	The value is an IP address.
prefix_bits	The number of prefix digits.

The varchar type.

Obtain the prefix of the IP address in the value of the client_ip field.

- Query statement

```
* | SELECT ip_prefix(client_ip,24) AS client_ip
```

- Query and analysis result

is_prefix_subnet_of function

The is_prefix_subnet_of function is used to check whether a CIDR block is a subnet of a specified CIDR block.

```
is_prefix_subnet_of(,)
```

Parameter	Description
	The value is a CIDR block. This function checks whether the y CIDR block is a subnet of the x CIDR block.
	The value is a CIDR block.

The Boolean type.

Check whether the value of the client_ip field is a subnet of 192.168.0.1/24.

- Query statement

```
* | SELECT is_prefix_subnet_of('192.168.0.1/24',concat(client_ip,'/24'))
```

- Query and analysis result

is_subnet_of function

The is_subnet_of function is used to check whether an IP address is in a specified CIDR block.

```
is_subnet_of(,)
```

Parameter	Description
	The value is a CIDR block.
	The value is an IP address.

The Boolean type.

Check whether the value of the client_ip field is in 192.168.0.1/24.

- Query statement

```
* | SELECT is_subnet_of('192.168.0.1/24',client_ip)
```

- Query and analysis result

ip_subnet_min function

The ip_subnet_min function is used to obtain the smallest IP address in a CIDR block.

```
ip_subnet_min()
```

Parameter	Description
	The value is a CIDR block.

The varchar type.

Obtain the smallest IP address in the CIDR block to which the value of the client_ip field belongs.

- Query statement

```
* | SELECT ip_subnet_min(concat(client_ip,'/24'))
```

- Query and analysis result

ip_subnet_max function

The ip_subnet_min function is used to obtain the largest IP address in a CIDR block.

```
ip_subnet_max()
```

vac

Parameter	Description
	The value is a CIDR block.

The varchar type.

Obtain the largest IP address in the CIDR block to which the value of the client_ip field belongs.

- Query statement

```
* | SELECT ip_subnet_max(concat(client_ip,'/24'))
```

- Query and analysis result

ip_subnet_range function

The ip_subnet_range function is used to obtain a CIDR block.

```
ip_subnet_range()
```

Parameter	Description
	The value is a CIDR block.

The JSON type.

Obtain the CIDR block to which the value of the client_ip field belongs.

- Query statement

```
* | SELECT ip_subnet_range(concat(client_ip,'/24'))
```

- Query and analysis result

8.1.16. URL functions

This topic describes the syntax of URL functions. This topic also provides examples on how to use the functions.

Notice

- The format of a URL is `[protocol:][//host[:port]][path][?query][#fragment]`.
- If you want to use strings in analytic statements, you must enclose strings in single quotation marks (''). Strings that are not enclosed or enclosed in double quotation marks ("") indicate field names or column names. For example, 'status' indicates the status string, and status or "status" indicates the status log field.

Function	Syntax	Description
url_encode function	url_encode()	Encodes a URL.
url_decode function	url_decode()	Decodes a URL.
url_extract_fragment function	url_extract_fragment()	Extracts the fragment from a URL.
url_extract_host function	url_extract_host()	Extracts the host from a URL.
url_extract_parameter function	url_extract_parameter(,*parameter name*)	Extracts the value of a specified parameter in the query string from a URL.
url_extract_path function	url_extract_path()	Extracts the path from a URL.
url_extract_port function	url_extract_port()	Extracts the port number from a URL.
url_extract_protocol function	url_extract_protocol()	Extracts the protocol from a URL.
url_extract_query function	url_extract_query()	Extracts the query string from a URL.

url_encode function

The url_encode function is used to encode a URL.

```
url_encode()
```

Parameter	Description
	The value of this parameter is a specific URL.

The varchar type.

Encode the value of the url field.

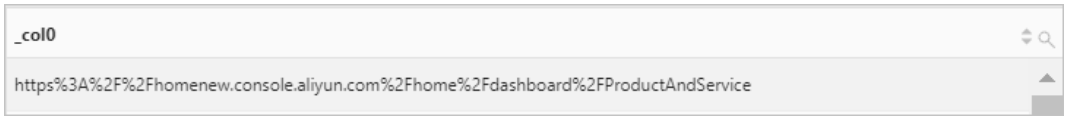
- Sample field

  ```
  url:https://homenew.console.aliyun.com/home/dashboard/ProductAndService
  ```

- Query statement

  ```
  * | select url_encode(url)
  ```

- Query and analysis result



url_decode function

The url_decode function is used to decode a URL.

```
url_decode()
```

Parameter	Description
	The value of this parameter is an encoded URL.

The varchar type.

Decode the value of the url field.

- Sample field

  ```
  url:http%3A%2F%2Fwww.aliyun.com%3A80%2Fproduct%2Fsls
  ```

- Query statement

  ```
  * | SELECT url_decode(url) AS decode
  ```

- Query and analysis result

url_extract_fragment function

The url_extract_fragment function is used to extract the fragment from a URL.

```
url_extract_fragment()
```

Parameter	Description
	The value of this parameter is a specific URL.

The varchar type.

Extract the fragment from the value of the url field.

- Sample field

  ```
  url:https://sls.console.aliyun.com/#/project/dashboard-demo/categoryList
  ```

- Query statement

  ```
  * | SELECT url_extract_fragment(url)
  ```

- Query and analysis result

url_extract_host function

The url_extract_host function is used to extract the host from a URL.

```
url_extract_host()
```

Parameter	Description
	The value of this parameter is a specific URL.

The varchar type.

Extract the host from the value of the url field.

- Sample field

  ```
  url:https://homenew.console.aliyun.com/home/dashboard/ProductAndService
  ```

- Query statement

  ```
  * | SELECT url_extract_host(url) AS host
  ```

- Query and analysis result

url_extract_parameter function

The url_extract_parameter function is used to extract the value of a specified parameter in the query string from a URL.

```
url_extract_parameter(,parameter name)
```

Parameter	Description
	The value of this parameter is a specific URL.
parameter name	The name of the parameter in the query string of the URL.

The varchar type.

Extract the value of the accounttraceid parameter from the value of the url field.

- Sample field

  ```
  url:https://sls.console.aliyun.com/lognext/project/dashboard-all/logsearch/nginx-demo?accounttrace
  id=d6241a173f88471c91d3405cda010ff5ghdw
  ```

- Query statement

  ```
  * | SELECT url_extract_parameter(url,'accounttraceid') AS accounttraceid
  ```

- Query and analysis result

url_extract_path function

The url_extract_path function is used to extract the path from a URL.

```
url_extract_path()
```

Parameter	Description
	The value of this parameter is a specific URL.

The varchar type.

Extract the path from the value of the url field.

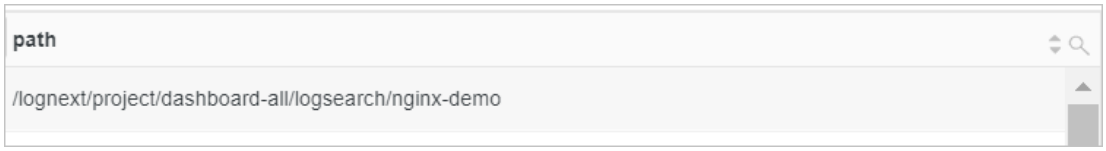
- Sample field

  ```
  url:https://sls.console.aliyun.com/lognext/project/dashboard-all/logsearch/nginx-demo?accounttrace
  id=d6241a173f88471c91d3405cda010ff5ghdw
  ```

- Query statement

  ```
  * | SELECT url_extract_path(url) AS path
  ```

- Query and analysis result

url_extract_port function

The url_extract_port function is used to extract the port number from a URL.

```
url_extract_port()
```

Parameter	Description
	The value of this parameter is a specific URL.

The varchar type.

Extract the port number from the value of the url field.

- Sample field

  ```
  url:http://localhost:8080/lognext/profile
  ```

- Query statement

  ```
  * | SELECT url_extract_port(url) AS port
  ```

- Query and analysis result

url_extract_protocol function

The url_extract_protocol function is used to extract the protocol from a URL.

```
url_extract_port()
```

Parameter	Description
	The value of this parameter is a specific URL.

The varchar type.

Extract the protocol from the value of the url field.

- Sample field

  ```
  url:https://homenew.console.aliyun.com/home/dashboard/ProductAndService
  ```

- Query statement

  ```
  * | SELECT url_extract_protocol(url) AS protocol
  ```

- Query and analysis result

url_extract_query function

The url_extract_query function is used to extract the query string from a URL.

```
url_extract_query()
```

Parameter	Description
	The value of this parameter is a specific URL.

The varchar type.

Extract the query string from the value of the url field.

- Sample field

  ```
  url:https://sls.console.aliyun.com/lognext/project/dashboard-all/logsearch/nginx-demo?accounttrace
  id=d6241a173f88471c91d3405cda010ff5ghdw
  ```

- Query statement

  ```
  * | SELECT url_extract_query(url)
  ```

- Query and analysis result

8.1.17. Approximate functions

This topic describes the syntax of approximate functions. This topic also provides examples on how to use the functions.

The following table describes the approximate functions that are supported by Log Service.

Notice

Function	Syntax	Description
approx_distinct function	approx_distinct()	Estimates the number of unique values in . The default standard error is 2.3%.
	approx_distinct(, *e*)	Estimates the number of unique values in . You can specify a custom standard error.
approx_percentile function	approx_percentile(, *percentage*)	Lists the values of in ascending order and returns the value that is approximately at the *percentage* position.
	approx_percentile(, array[*percentage01*, *percentage02*...])	Lists the values of in ascending order and returns the values that are approximately at the *percentage01* and *percentage02* positions.
	approx_percentile(, *weight*, *percentage*)	Calculates the product of each value and the weight of the value, sorts all values in ascending order of the calculated products, and then returns the value that is approximately at the *percentage* position.
	approx_percentile(, *weight*, array[*percentage01*, *percentage02*...])	Calculates the product of each value and the weight of the value, sorts all values in ascending order of the calculated products, and then returns the values that are approximately at the *percentage01* and *percentage02* positions.
	approx_percentile(, *weight*, *percentage*, *accuracy*)	Calculates the product of each value and the weight of the value, sorts all values in ascending order of the calculated products, and then returns the value that is approximately at the *percentage* position. You can specify the accuracy of the return value.
numeric_histogram function	numeric_histogram(*bucket*,)	Computes the approximate histogram of based on the number of histogram columns. The number is specified in the bucket parameter. The return value is of the JSON type.
	numeric_histogram(*bucket*, , *weight*)	Computes the approximate histogram of based on the number of histogram columns. The number is specified in the bucket parameter. The return value is of the JSON type. You can specify weights for the values of .

Function	Syntax	Description
numeric_histogram_u function	numeric_histogram_u(*bucket*,)	Computes the approximate histogram of based on the number of histogram columns. The number is specified in the bucket parameter. A table that contains multiple rows and columns is returned.

approx_distinct function

The approx_distinct function is used to estimate the number of unique values in .

- An approx_distinct function of the following syntax is used to estimate the number of unique values in . The default standard error is 2.3%.

```
approx_distinct()
```

- An approx_distinct function of the following syntax is used to estimate the number of unique values in . You can specify a custom standard error.

```
approx_distinct(, e)
```

Parameter	Description
	The value of this parameter is of an arbitrary data type.
e	The custom standard error. Valid values: 0.0115 to 0.26.

The bigint type.

- Example 1: Use the count function to calculate the number of page views (PVs). Then, use the approx_distinct function to estimate the unique values of the client_ip field as the number of unique visitors (UVs). The standard error is 2.3%.
  - Query statement

```
* |SELECT count(*) AS PV, approx_distinct(client_ip) AS UV
```

  - Query and analysis result

PV	UV
941787	723040

- Example 2: Use the count function to calculate the number of PVs. Then, use the approx_distinct function to estimate the unique values of the client_ip field as the number of UVs. The standard error is 10%.
  - Query statement

```
* |SELECT count(*) AS PV, approx_distinct(client_ip,0.1) AS UV
```

  - Query and analysis result

PV	UV
9095	7946

approx_percentile function

The approx_percentile function is used to list the values of in ascending order and return the value that is approximately at the *percentage* position.

- An approx_percentile function of the following syntax is used to list the values of in ascending order and return the value that is approximately at the *percentage* position. The return value is of the double type.

```
approx_percentile(, percentage)
```

- An approx_percentile function of the following syntax is used to list the values of in ascending order and return the values that are approximately at the percentage01 and percentage02 positions. The return value is of the array(double,double) type.

```
approx_percentile(, array[percentage01, percentage02...])
```

- An approx_percentile function of the following syntax is used to calculate the product of each value and the weight of the value, sort all values in ascending order of the calculated products, and then return the value that is approximately at the *percentage* position. The return value is of the double type.

```
approx_percentile(, weight, percentage)
```

- An approx_percentile function of the following syntax is used to calculate the product of each value and the weight of the value, sort all values in ascending order of the calculated products, and then return the values that are approximately at the percentage01 and percentage 02 positions. The return value is of the array(double,double) type.

```
approx_percentile(, weight, array[percentage01, percentage02...])
```

- An approx_percentile function of the following syntax is used to calculate the product of each value and the weight of the value, sort all values in ascending order of the calculated products, and then return the value that is approximately at the *percentage* position. The return value is of the double type. You can specify the accuracy of the return value.

```
approx_percentile(, weight, percentage, accuracy)
```

Parameter	Description
	The value of this parameter is of the double type.
percentage	The percentage value. Value range: [0, 1].
accuracy	The accuracy. Value range: (0, 1).
weight	The weight. A weight must be an integer that is greater than 1. After you specify weights, the system calculates the product of each value and the weight of the value and sorts all values in ascending order of the calculated products.

The double or array(double,double) type.

- Example 1: Sort the values of the request_time column in ascending order, and then return the value that is approximately at the 50% position in the request_time field.
  - Query statement

```
*| SELECT approx_percentile(request_time,0.5)
```

  - Query and analysis result

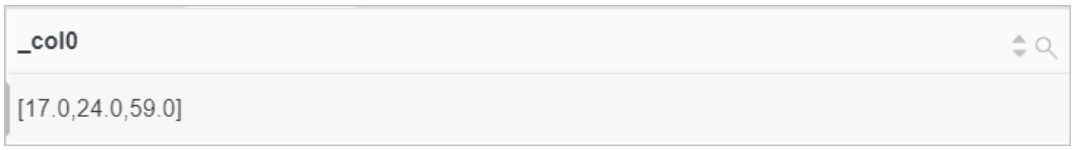
- Example 2: Sort the values of the request_time column in ascending order, and then return the values that are

approximately at the 10%, 20%, and 70% positions in the request_time field.

- Query statement

```
*| SELECT approx_percentile(request_time,array[0.1,0.2,0.7])
```

- Query and analysis result



- Example 3: Calculate the product of each request_time value and the weight of the value, sort all request_time values in ascending order of the calculated products, and then return the value that is approximately at the 50% position in the request_time field. If the value of request_time is less than 20, the weight is 100. Otherwise, the weight is 10.
  - Query statement

```
* |
SELECT
  approx_percentile(
    request_time,case
      when request_time < 20 then 100
      else 10
    end,
    0.5
  )
```

  - Query and analysis result

- Example 4: Calculate the product of each request_time value and the weight of the value, sort all request_time values in ascending order of the calculated products, and then return the values that are approximately at the 80% and 90% positions in the request_time field. If the value of request_time is less than 20, the weight is 100. Otherwise, the weight is 10.
  - Query statement

```
* |
SELECT
  approx_percentile(
    request_time,case
      when request_time < 20 then 100
      else 10
    end,
    array [0.8,0.9]
  )
```

  - Query and analysis result

- Example 5: Calculate the product of each request_time value and the weight of the value, sort all request_time values in ascending order of the calculated products, and then return the value that is approximately at the 50% position in the request_time field. The accuracy is 0.2. If the value of request_time is less than 20, the

weight is 100. Otherwise, the weight is 10.

- Query statement

```
* |
SELECT
  approx_percentile(
    request_time,case
      when request_time < 20 then 100
      else 10
    end,
    0.5,
    0.2
  )
```

- Query and analysis result

_col0
18.0

numeric_histogram function

The numeric_histogram function is used to compute the approximate histogram of based on the number of histogram columns. The number is specified in the bucket parameter. The return value is of the JSON type.

- A numeric_histogram function of the following syntax is used to compute the approximate histogram of based on the number of histogram columns. The number is specified in the bucket parameter.

```
numeric_histogram(bucket, )
```

- A numeric_histogram function of the following syntax is used to compute the approximate histogram of based on the number of histogram columns. The number is specified in the bucket parameter. You can specify weights for the values of .

```
numeric_histogram(bucket, , weight)
```

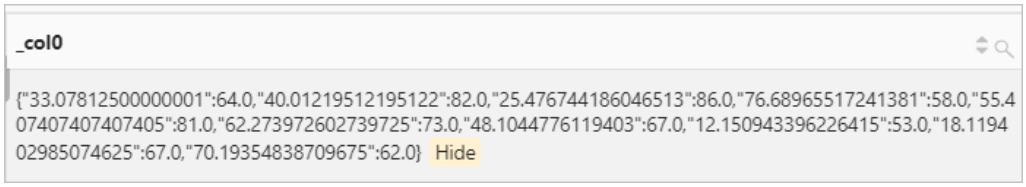
Parameter	Description
bucket	The number of columns in the histogram. The value of this parameter is of the bigint type.
	The value of this parameter is of the double type.
weight	The weight. A weight must be an integer that is greater than 0. After you specify weights, the system calculates the product of each value and the weight of the value and groups the values based on the calculated products.

The JSON type.

- Example 1: Compute the approximate histogram of the request duration for the POST method.
  - Query statement

```
request_method:POST | SELECT numeric_histogram(10,request_time)
```

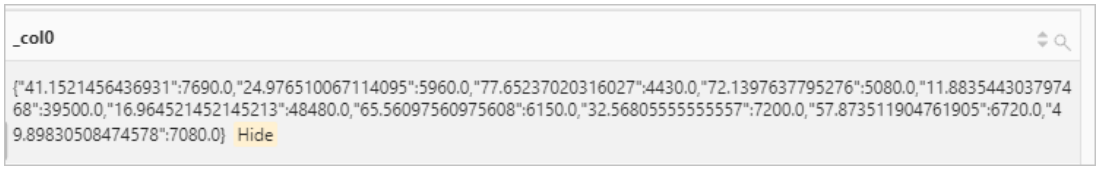
- Query and analysis result



- Example 2: Calculate the product of each request_time value and the weight of the value, group the values based on the calculated products, and then compute the approximate histogram of the request duration for the POST method.
  - Query statement

```
request_method:POST| SELECT numeric_histogram(10, request_time,case when request_time<20 then 10
0 else 10 end)
```

  - Query and analysis result



numeric_histogram_u function

The numeric_histogram_u function is used to compute the approximate histogram of based on the number of histogram columns. The number is specified in the bucket parameter. A table that contains multiple rows and columns is returned.

```
numeric_histogram_u(bucket, )
```

Parameter	Description
bucket	The number of columns in the histogram. The value of this parameter is of the bigint type.
	The value of this parameter is of the double type.

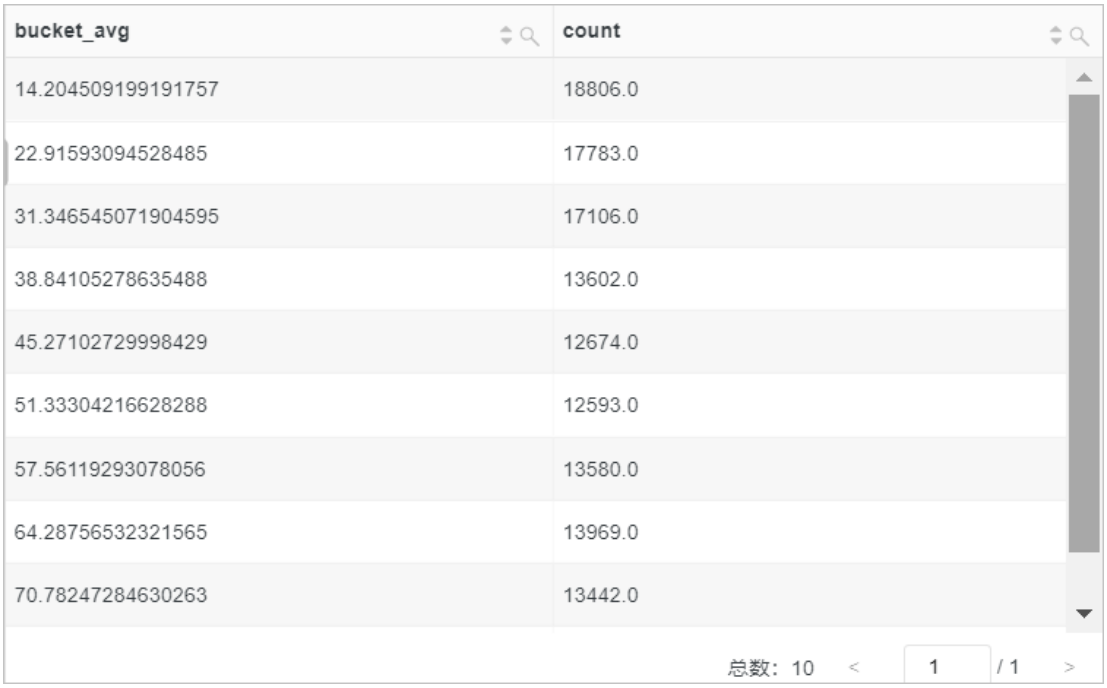
The double type.

Compute the approximate histogram of the request duration for the POST method.

- Query statement

```
request_method:POST | select numeric_histogram_u(10,request_time)
```

- Query and analysis result



bucket_avg	count
14.204509199191757	18806.0
22.91593094528485	17783.0
31.346545071904595	17106.0
38.84105278635488	13602.0
45.27102729998429	12674.0
51.33304216628288	12593.0
57.56119293078056	13580.0
64.28756532321565	13969.0
70.78247284630263	13442.0

总数：10 < 1 / 1 >

8.1.18. Binary functions

This topic describes the syntax of binary functions. This topic also provides examples on how to use the functions.

The following table describes the binary functions that are supported by Log Service.

Notice

- If you want to use strings in analytic statements, you must enclose the strings in single quotation marks (''). Strings that are not enclosed or enclosed in double quotation marks ("") indicate field names or column names. For example, 'status' indicates the status string, and status or "status" indicates the status log field.
- varbinary is a binary character type, and varchar is a variable-length character type.

Function	Syntax	Description
from_base64 function	from_base64()	Converts a Base64-encoded string to a binary number.
from_base64url function	from_base64url()	Converts a Base64-encoded string to a binary number by using URL reserved characters.
from_big_endian_64 function	from_big_endian_64()	Converts a binary number in big endian mode to a bigint value.
from_hex function	from_hex()	Converts a hexadecimal number to a binary number.
length function	length()	Calculates the length of a binary number.
md5 function	md5()	Computes the MD5 hash value for a binary number.

Function	Syntax	Description
to_base64 function	to_base64()	Converts a binary number to a Base64-encoded string.
to_base64url function	to_base64url()	Converts a binary number to a Base64-encoded string by using URL reserved characters.
to_hex function	to_hex()	Converts a binary number to a hexadecimal number.
to_big_endian_64 function	to_big_endian_64()	Converts a bigint value to a binary number in big endian mode.
sha1 function	sha1()	Computes the SHA-1 hash value for a binary number.
sha256 function	sha256()	Computes the SHA-256 hash value for a binary number.
sha512 function	sha512()	Computes the SHA-512 hash value for a binary number.
xxhash64 function	xxhash64()	Computes the xxhash64 hash value for a binary number.

from_base64 function

The from_base64 function is used to convert a Base64-encoded string to a binary number.

```
from_base64()
```

Parameter	Description
	The value of this parameter is of the binary type.

The varbinary type.

Notice The return value of the varbinary type contains invisible characters and is displayed in the Base64-encoded format.

- If the returned binary number is an invisible character, you can use the to_hex function to convert the number to a hexadecimal number.
- If the returned binary number is a visible character, you can use the from_utf8 function to convert the number to a UTF-8 string.

Convert a Base64-encoded string to a binary number and then convert the binary number to a hexadecimal number.

- Query statement

```
* | SELECT to_hex(from_base64('c2xz'))
```

- Query and analysis result

from_base64url function

The from_base64url function is used to convert a Base64-encoded string to a binary number by using URL reserved characters.

```
from_base64url()
```

Parameter	Description
	The value of this parameter is of the binary type.

The varbinary type.

Notice The return value of the varbinary type contains invisible characters and is displayed in the Base64-encoded format.

- If the returned binary number is an invisible character, you can use the to_hex function to convert the number to a hexadecimal number.
- If the returned binary number is a visible character, you can use the from_utf8 function to convert the number to a UTF-8 string.

Convert a Base64-encoded string to a binary number by using URL reserved characters.

- Query statement

```
* | SELECT to_hex(from_base64url('c2xz'))
```

- Query and analysis result

from_big_endian_64 function

The from_big_endian_64 function is used to convert a binary number in big endian mode to a bigint value.

```
from_big_endian_64()
```

Parameter	Description
	The value of this parameter is of the binary type.

The bigint type.

Convert the binary number 10 in big endian mode to a bigint value.

- Query statement

```
* | SELECT from_big_endian_64(to_big_endian_64(10))
```

- Query and analysis result

from_hex function

The from_hex function is used to convert a hexadecimal number to a binary number.

```
from_hex()
```

Parameter	Description
	The value of this parameter is of the varbinary type.

The varbinary type.

Convert the hexadecimal number D74D to a binary number.

- Query statement

```
* | SELECT from_hex('D74D')
```

- Query and analysis result

length function

The length function is used to calculate the length of a binary number.

```
length()
```

Parameter	Description
	The value of this parameter is of the binary type.

The bigint type.

Calculate the length of the region field value.

- Query statement

```
* | SELECT length('00101000')
```

- Query and analysis result

md5 function

The md5 function is used to compute the MD5 hash value for a binary number.

```
md5()
```

Parameter	Description
	The value of this parameter is of the varbinary type.

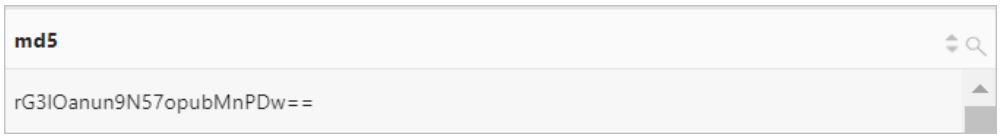
The varbinary type.

Compute the MD5 hash value for the binary number 1101.

- Query statement

```
* | SELECT MD5(from_base64('1101')) AS md5
```

- Query and analysis result



to_base64 function

The to_base64 function is used to convert a binary number to a Base64-encoded string.

```
to_base64()
```

Parameter	Description
	The value of this parameter is of the binary type.

The varchar type.

Convert the binary number 10 to a Base64-encoded string.

- Query statement

```
* | SELECT  to_base64(from_base64('10')) AS base64
```

- Query and analysis result

to_base64url function

The to_base64url function is used to convert a binary number to a Base64-encoded string by using URL reserved characters.

```
to_base64url()
```

Parameter	Description
	The value of this parameter is of the binary type.

The varchar type.

Convert the binary number 100 to a Base64-encoded string by using URL reserved characters.

- Query statement

```
* | SELECT  to_base64url(from_base64('100'))
```

- Query and analysis result

to_hex function

The to_hex function is used to convert a binary number to a hexadecimal number.

```
to_hex()
```

Parameter	Description
	The value of this parameter is of the binary type.

The varchar type.

Convert the binary number 100 to a hexadecimal number.

- Query statement

```
* | SELECT to_hex(from_base64('100'))
```

- Query and analysis result

to_big_endian_64 function

The to_big_endian_64 function is used to convert a bigint value to a binary number in big endian mode.

```
to_big_endian_64()
```

Parameter	Description
	The value of this parameter is of the bigint type.

The varbinary type.

Convert the bigint value 0 to a binary number in big endian mode.

- Query statement

```
* | SELECT to_big_endian_64(0)
```

- Query and analysis result

sha1 function

The sha1 function is used to compute the SHA-1 hash value for a binary number.

```
sha1()
```

Parameter	Description
	The value of this parameter is of the binary type.

The varbinary type.

Compute the SHA-1 hash value for the binary number 1101.

- Query statement

```
* | SELECT sha1(from_base64('1101')) AS sha1
```

- Query and analysis result

sha256 function

The sha256 function is used to compute the SHA-256 hash value for a binary number.

```
sha256()
```

Parameter	Description
	The value of this parameter is of the binary type.

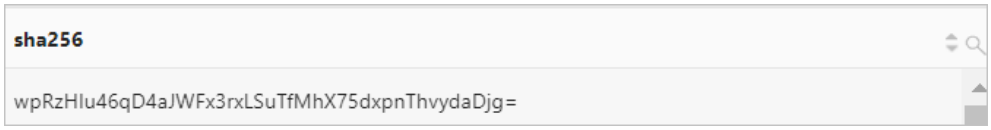
The varbinary type.

Compute the SHA-256 hash value for the binary number 1101.

- Query statement

```
* | SELECT sha256(from_base64('1101')) AS sha256
```

- Query and analysis result



sha512 function

The sha512 function is used to compute the SHA-512 hash value for a binary number.

```
sha512()
```

Parameter	Description
	The value of this parameter is of the binary type.

The varbinary type.

Compute the SHA-512 hash value for the binary number 1101.

- Query statement

```
* | SELECT sha512(from_base64('1101')) AS sha512
```

- Query and analysis result

xxhash64 function

The xxhash64 function is used to compute the xxhash64 hash value for a binary number.

```
xxhash64()
```

Parameter	Description
	The value of this parameter is of the binary type.

The varbinary type.

Compute the xxhash64 hash value for the binary number 10.

- Query statement

```
* | SELECT xxhash64(from_base64('10'))
```

- Query and analysis result

8.1.19. Bitwise functions

This topic describes the syntax of bitwise functions. This topic also provides examples on how to use the functions.

Notice

Function	Syntax	Description
bit_count function	bit_count(,*bits*)	Returns the number of bits 1 in in binary representation.
bitwise_and function	bitwise_and(,)	Returns the result of the bitwise AND operation on and in binary representation.
bitwise_not function	bitwise_not()	Returns the result of the bitwise NOT operation on in binary representation.
bitwise_or function	bitwise_or(,)	Returns the result of the bitwise OR operation on and in binary representation.

Function	Syntax	Description
bitwise_xor function	bitwise_xor(,)	Returns the result of the bitwise XOR operation on and in binary representation.

bit_count function

The bit_count function returns the number of bits 1 in x in binary representation.

```
bit_count(,bits)
```

Parameter	Description
	The value of this parameter is of the bigint type.
bits	The value of this parameter is the number of bits, such as 64 bits.

The bigint type.

Convert the number 24 into a binary number and obtain the number of bits 1 in the binary number.

- Query statement

```
* | SELECT bit_count(24, 64)
```

- Query and analysis results

bitwise_and function

The bitwise_and function returns the result of the bitwise AND operation on and in binary representation.

```
bitwise_and(,)
```

Parameter	Description
	The value of this parameter is of the bigint type.
	The value of this parameter is of the bigint type.

The bigint type.

Perform a bitwise AND operation on 3 and 5 in binary representation.

- Query statement

```
* | SELECT bitwise_and(3, 5)
```

- Query and analysis results

bitwise_not function

The bitwise_not function returns the result of the bitwise NOT operation on in binary representation.

```
bitwise_not()
```

Parameter	Description
	The value of this parameter is of the bigint type.

The bigint type.

Perform a bitwise NOT operation on 4 in binary representation.

- Query statement

```
* | SELECT bitwise_not(4)
```

- Query and analysis results

bitwise_or function

The bitwise_or function returns the result of the bitwise OR operation on and in binary representation.

```
bitwise_or(,)
```

Parameter	Description
	The value of this parameter is of the bigint type.
	The value of this parameter is of the bigint type.

The bigint type.

Perform a bitwise OR operation on 3 and 5 in binary representation.

- Query statement

```
* | SELECT bitwise_or(3, 5)
```

- Query and analysis results

bitwise_xor function

The bitwise_xor function returns the result of the bitwise XOR operation on and in binary representation.

```
bitwise_xor(,)
```

Parameter	Description
	The value of this parameter is of the bigint type.

Parameter	Description
	The value of this parameter is of the bigint type.

The bigint type.

Perform a bitwise XOR operation on 3 and 5 in binary representation.

- Query statement

```
?1* | SELECT bitwise_xor(3, 5)
```

- Query and analysis results

8.1.20. Geospatial functions

This topic describes the syntax of geospatial functions. This topic also provides examples on how to use geospatial functions.

Introduction

Geospatial functions that start with the ST_ prefix comply with the SQL/MM standard and the OpenGIS Abstract Specification of the Open Geospatial Consortium (OGC). Geospatial functions use well-known text (WKT) representations to describe geometries, such as points, line strings, and polygons. The following table describes the geometries and the WKT representations that are used to describe the geometries.

Geometry	WKT representation
Point	POINT (0 0)
Line string	LINESTRING (0 0,1 1,1 2)
Polygon	POLYGON((0 0,4 0,4 4,0 4,0 0), (1 1,2 1,2 2,1 2,1 1))
Multipoint	MULTIPOINT(0 0,1 2)
Multilinestring	MULTILINESTRING((0 0,1 1,1 2), (2 3,3 2,5 4))
Multipolygon	MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0), (1 1,2 1,2 2,1 2,1 1)), ((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))
Geometry collection	GEOMETRYCOLLECTION(POINT(2 3), LINESTRING(2 3,3 4))

Functions

Type	Function	Syntax	Description
Constructors	ST_AsText function	ST_AsText()	Returns the WKT representation of a geometry.
	ST_GeometryFromText function	ST_GeometryFromText()	Returns a geometry from the specified WKT representation.
	ST_LineFromText function	ST_LineFromText()	Returns a line string from the specified WKT representation.

Type	Function	Syntax	Description
	ST_Polygon function	ST_Polygon()	Returns a polygon from the specified WKT representation.
	ST_Point function	ST_Point(,)	Returns a point from the specified WKT representation.
Operators	ST_Boundary function	ST_Boundary()	Returns the closure of the combinatorial boundary of a geometry.
	ST_Buffer function	ST_Buffer(, *distance*)	Returns a geometry that represents all points whose distance from the specified geometry is less than or equal to the specified distance.
	ST_Difference function	ST_Difference(,)	Returns a geometry that represents the point set difference of two specified geometries.
	ST_Envelope function	ST_Envelope()	Returns the bounding rectangular polygon of a geometry.
	ST_ExteriorRing function	ST_ExteriorRing()	Returns a line string that represents the exterior ring of a geometry.
	ST_Intersection function	ST_Intersection(,)	Returns a geometry that represents the point set intersection of two specified geometries.
	ST_SymDifference function	ST_SymDifference(,)	Returns a geometry that represents the point set symmetric difference of two specified geometries.
Spatial relationship tests	ST_Contains function	ST_Contains(,)	Returns true if no points of the second geometry lie in the exterior of the first geometry and at least one point of the interior of the first geometry lies in the interior of the second geometry.
	ST_Crosses function	ST_Crosses(,)	Returns true if two specified geometries have several interior points in common.
	ST_Disjoint function	ST_Disjoint(,)	Returns true if two specified geometries do not share any portion of two-dimensional space.
	ST_Equals function	ST_Equals(,)	Returns true if two specified geometries represent the same geometry.
	ST_Intersects function	ST_Intersects(,)	Returns true if two specified geometries share a portion of two-dimensional space.

Type	Function	Syntax	Description
	ST_Overlaps function	ST_Overlaps(,)	Returns true if two specified geometries share space and have the same dimension but are not completely contained by each other.
	ST_Relate function	ST_Relate(,,*patternMatrix string*)	Returns true if two specified geometries have a spatial relationship.
	ST_Touches function	ST_Touches(,)	Returns true if two specified geometries have at least one point in common but their interiors do not intersect.
	ST_Within function	ST_Within(,)	Returns true if the first geometry is completely inside the second geometry.
Accessors	ST_Area function	ST_Area()	Calculates the projected area of a geometry on a two-dimensional plane by using the Euclidean distance method.
	ST_Centroid function	ST_Centroid()	Returns the point value that represents the mathematical centroid of a geometry.
	ST_CoordDim function	ST_CoordDim()	Returns the coordinate dimension of a geometry.
	ST_Dimension function	ST_Dimension()	Returns the inherent dimension of a geometry. The inherent dimension must be less than or equal to the coordinate dimension.
	ST_Distance function	ST_Distance(,)	Returns the minimum distance between two geometries.
	ST_EndPoint function	ST_EndPoint()	Returns the last point of a line string.
	ST_IsClosed function	ST_IsClosed()	Returns true if the start point of a line string coincides with the end point.
	ST_IsEmpty function	ST_IsEmpty()	Returns true if a geometry is empty.
	ST_IsRing function	ST_IsRing()	Returns true if a line string is closed and simple.
	ST_Length function	ST_Length()	Calculates the projected length of a line string on a two-dimensional plane by using the Euclidean distance method. If multiple line strings exist, the function returns the sum of the lengths of the multiple line strings.

Type	Function	Syntax	Description
	ST_NumPoints function	ST_NumPoints()	Returns the number of points in a geometry.
	ST_NumInteriorRing function	ST_NumInteriorRing()	Returns the number of interior rings in a geometry.
	ST_StartPoint function	ST_StartPoint()	Returns the first point of a line string.
	ST_X function	ST_X()	Returns the X-coordinate of a specified point.
	ST_XMax function	ST_XMax()	Returns the maximum first X-coordinate of a geometry.
	ST_XMin function	ST_XMin()	Returns the minimum first X-coordinate of a geometry.
	ST_Y function	ST_Y()	Returns the Y-coordinate of a specified point.
	ST_YMax function	ST_YMax()	Returns the maximum first Y-coordinate of a geometry.
	ST_YMin function	ST_YMin()	Returns the minimum first Y-coordinate of a geometry.
Bing tiles	bing_tile function	bing_tile(, , *zoom_level*)	Returns a Bing tile based on the X-coordinate, Y-coordinate, and zoom level.
		bing_tile(*quadKey*)	Returns a Bing tile based on the quadtree key.
	bing_tile_at function	bing_tile_at(,,*zoom_level*)	Returns a Bing tile based on the latitude, longitude, and zoom level.
	bing_tile_coordinates function	bing_tile_coordinates()	Returns the X- and Y-coordinates of a Bing tile.
	bing_tile_polygon function	bing_tile_polygon()	Returns the polygon format of a Bing tile.
	bing_tile_quadkey function	bing_tile_quadkey()	Returns the quadtree key of a Bing tile.
	bing_tile_zoom_level function	bing_tile_zoom_level()	Returns the zoom level of a Bing tile.

ST_AsText function

The ST_AsText function returns the WKT representation of a geometry.

```
ST_AsText()
```

Parameter	Description

Parameter	Description
	The value of this parameter is of the geometry type.

The varchar type.

Obtain the WKT representation of a point.

- Query statement

```
* | SELECT ST_AsText(ST_Point(1,1))
```

- Query and analysis results

ST_GeometryFromText function

The ST_GeometryFromText function returns a geometry from the WKT representation that you specify.

```
ST_GeometryFromText()
```

Parameter	Description
	The value of this parameter is of the varchar type.

The geometry type.

Construct multiple polygons.

- Query statement

```
* | SELECT ST_GeometryFromText('multipolygon(((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,
60 40,50 40)))')
```

- Query and analysis results

ST_LineFromText function

The ST_LineFromText function returns a line string from the WKT representation that you specify.

```
ST_LineFromText()
```

Parameter	Description
	The value of this parameter is of the varchar type.

The linestring type.

Construct a line string.

- Query statement

```
* | SELECT ST_LineFromText('linestring(10 10,20 20)')
```

- Query and analysis results

ST_Polygon function

The ST_Polygon function returns a polygon from the WKT representation that you specify.

```
ST_Polygon()
```

Parameter	Description
	The value of this parameter is of the varchar type.

The polygon type.

Construct a polygon.

- Query statement

```
* | SELECT ST_Polygon('polygon((10 10,10 20,20 20,20 15,10 10))')
```

- Query and analysis results

ST_Point function

The ST_Point function returns a point from the WKT representation that you specify.

```
ST_Point(,)
```

Parameter	Description
	The value of this parameter is of the geometry type.
	The value of this parameter is of the geometry type.

The point type.

Construct a point.

- Query statement

```
* | SELECT ST_Point(0,0)
```

- Query and analysis results

ST_Boundary function

The ST_Boundary function returns the closure of the combinatorial boundary of a geometry.

- The closure of the combinatorial boundary of a point is empty. If the geometry that you specify is a point, the function returns POINT EMPTY.
- The closure of the combinatorial boundary of a line string is composed of the end points of the line string.
- The closure of the combinatorial boundary of a polygon is composed of line strings, including the exterior and interior rings of the polygon.

```
ST_Boundary()
```

Parameter	Description
	The value of this parameter is of the geography type.

The geography type.

Use the ST_Polygon function to return a polygon. Then, use the ST_Boundary function to return the closure of the combinatorial boundary of the polygon.

- Query statement

```
* | SELECT  ST_Boundary(ST_Polygon('polygon((10 10,10 20,20 20,20 15,10 10))'))
```

- Query and analysis results

ST_Buffer function

The ST_Buffer function returns a geometry that represents all points whose distance from the specified geometry is less than or equal to the specified distance.

```
ST_Buffer(,distance)
```

Parameter	Description
	The value of this parameter is of the geometry type.
distance	The distance.

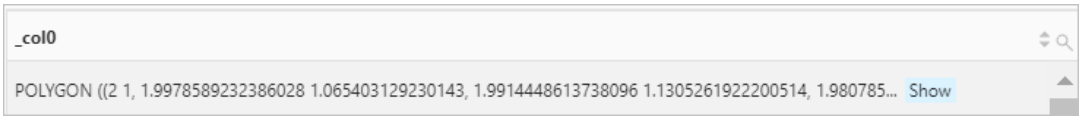
The geometry type.

Use the ST_Point function to return a point. Then, use the ST_Buffer function to return a polygon that represents all points whose distance from the point is less than or equal to the specified distance.

- Query statement

```
* | SELECT ST_Buffer(ST_Point(1,1),1)
```

- Query and analysis results



ST_Difference function

The ST_Difference function returns a geometry that represents the point set difference of two specified geometries.

```
ST_Difference(,)
```

Parameter	Description
	The value of this parameter is of the geometry type.
	The value of this parameter is of the geometry type.

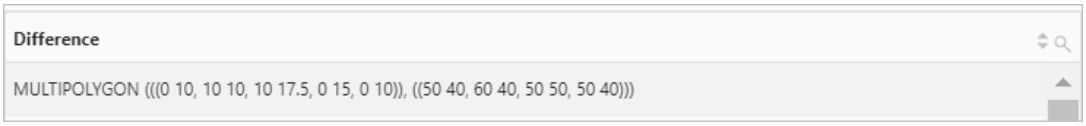
The geometry type.

Use the ST_GeometryFromText function to return two geometries. Then, use the ST_Difference function to return a geometry that represents the point set difference of the two geometries.

- Query statement

```
* |
SELECT
  ST_Difference(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,0 15,0 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    )
  ) AS "Difference"
```

- Query and analysis results



ST_Envelope function

The ST_Envelope function returns the bounding rectangular polygon of a geometry.

```
ST_Envelope()
```

Parameter	Description
	The value of this parameter is of the geometry type.

The geometry type.

Use the ST_GeometryFromText function to return a geometry. Then, use the ST_Envelope function to return the bounding rectangular polygon of the geometry.

- Query statement

```
* |
SELECT
  ST_Envelope(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- Query and analysis results

ST_ExteriorRing function

The ST_ExteriorRing function returns a line string that represents the exterior ring of a geometry.

```
ST_ExteriorRing()
```

Parameter	Description
	The value of this parameter is of the geometry type.

The geometry type.

Use the ST_GeometryFromText function to return a geometry. Then, use the ST_ExteriorRing function to return a line string that represents the exterior ring of the geometry.

- Query statement

```
* |
SELECT
  ST_ExteriorRing(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- Query and analysis results

ST_Intersection function

The ST_Intersection function returns a geometry that represents the point set intersection of two specified geometries.

```
ST_Intersection(,)
```

Parameter	Description
	The value of this parameter is of the geometry type.
	The value of this parameter is of the geometry type.

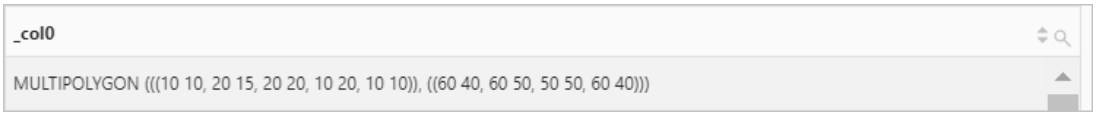
The geometry type.

Use the ST_GeometryFromText function to return two geometries. Then, use the ST_Intersection function to return a geometry that represents the point set intersection of the two geometries.

- Query statement

```
* |
SELECT
  ST_Intersection(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    )
  )
```

- Query and analysis results



ST_SymDifference function

The ST_SymDifference function returns a geometry that represents the point set symmetric difference of two specified geometries.

```
ST_SymDifference(,)
```

Parameter	Description
	The value of this parameter is of the geometry type.
	The value of this parameter is of the geometry type.

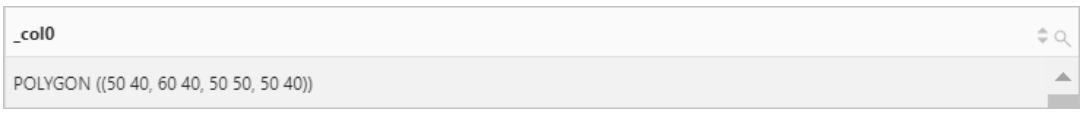
The geometry type.

Use the ST_GeometryFromText function to return two geometries. Then, use the ST_SymDifference function to return a geometry that represents the point set symmetric difference of the two geometries.

- Query statement

```
* |
SELECT
  ST_SymDifference(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    )
  )
```

- Query and analysis results



ST_Contains function

The ST_Contains function checks whether no points of the second geometry lie in the exterior of the first geometry and at least one point of the interior of the first geometry lies in the interior of the second geometry. If yes, the function returns true.

```
ST_Contains(,)
```

Parameter	Description
	The value of this parameter is of the geometry type.
	The value of this parameter is of the geometry type.

The Boolean type.

Use the ST_GeometryFromText function to return two geometries. Then, use the ST_Contains function to check whether no points of the second geometry lie in the exterior of the first geometry and at least one point of the interior of the first geometry lies in the interior of the second geometry.

- Query statement

```
* |
SELECT
  ST_Contains(
    ST_GeometryFromText(
      'polygon((10 10,10 20,20 20,20 15,10 10))'
    ),
    ST_GeometryFromText(
      'point(11 11)'
    )
  )
```

- Query and analysis results

ST_Crosses function

The ST_Crosses function checks whether two specified geometries have several interior points in common. If yes, the function returns true.

```
ST_Crosses(,)
```

Parameter	Description
	The value of this parameter is of the geometry type.
	The value of this parameter is of the geometry type.

The Boolean type.

Use the ST_GeometryFromText function to return two geometries. Then, use the ST_Crosses function to check whether the two geometries have several interior points in common.

- Query statement

```
* |
SELECT
  ST_Crosses(
    ST_GeometryFromText(
      'multipolygon (((10 10, 10 20, 20 20, 20 15 , 10 10), (50 40, 50 50, 60 50, 60 40, 50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10, 10 20, 20 20, 20 15 , 10 10), (50 40, 50 50, 60 50, 60 40, 50 50)))'
    )
  )
```

- Query and analysis results

ST_Disjoint function

The ST_Disjoint function checks whether two specified geometries do not share any portion of two-dimensional space. If yes, the function returns true.

```
ST_Disjoint(,)
```

Parameter	Description
	The value of this parameter is of the geometry type.
	The value of this parameter is of the geometry type.

The Boolean type.

Use the ST_GeometryFromText function to return two geometries. Then, use the ST_Disjoint function to check whether the two geometries do not share any portion of two-dimensional space.

- Query statement

```
* |
SELECT
   ST_Disjoint(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    )
  )
```

- Query and analysis results

ST_Equals function

The ST_Equals function checks whether two specified geometries represent the same geometry. If yes, the function returns true.

```
ST_Equals(,)
```

Parameter	Description
	The value of this parameter is of the geometry type.
	The value of this parameter is of the geometry type.

The Boolean type.

Use the ST_GeometryFromText function to return two geometries. Then, use the ST_Equals function to check whether the two geometries represent the same geometry.

- Query statement

```
* |
SELECT
   ST_Equals(
    ST_GeometryFromText(
      'multipolygon(((10 10,10 20,20 20,20 15,10 10),(50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon(((10 10,10 20,20 20,20 15,10 10),(50 40,50 50,60 50,60 40,50 50)))'
    )
  )
```

- Query and analysis results

ST_Intersects function

The ST_Intersects function checks whether two specified geometries share a portion of two-dimensional space. If yes, the function returns true.

```
ST_Intersects(,)
```

Parameter	Description
	The value of this parameter is of the geometry type.
	The value of this parameter is of the geometry type.

The Boolean type.

Use the ST_GeometryFromText function to return two geometries. Then, use the ST_Intersects function to check whether the two geometries share a portion of two-dimensional space.

- Query statement

```
* |
SELECT
   ST_Intersects(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    )
  )
```

- Query and analysis results

ST_Overlaps function

The ST_Overlaps function checks whether two specified geometries share space and have the same dimension but are not completely contained by each other. If yes, the function returns true.

```
ST_Overlaps(,)
```

Parameter	Description
	The value of this parameter is of the geometry type.
	The value of this parameter is of the geometry type.

The Boolean type.

Use the ST_GeometryFromText function to return two geometries. Then, use the ST_Overlaps function to check whether the two geometries share space and have the same dimension but are not completely contained by each other.

- Query statement

```
* |
SELECT
  ST_Overlaps(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    )
  )
```

- Query and analysis results

ST_Relate function

The ST_Relate function checks whether two specified geometries have a spatial relationship. If yes, the function returns true.

```
ST_Relate(,,patternMatrix string)
```

Parameter	Description
	The value of this parameter is of the geometry type.
	The value of this parameter is of the geometry type.
patternMatrix string	The DE-9IM pattern matrix string. The value of this parameter is of the varchar type.

The Boolean type.

Use the ST_GeometryFromText function to return two geometries. Then, use the ST_Relate function to check whether the two geometries have a spatial relationship.

- Query statement

```
* |
SELECT
  ST_Relate(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    ),  '****T****'
  )
```

- Query and analysis results

ST_Touches function

The ST_Touches function checks whether two specified geometries have at least one point in common but their interiors do not intersect. If yes, the function returns true.

```
ST_Touches(,)
```

Parameter	Description
	The value of this parameter is of the geometry type.
	The value of this parameter is of the geometry type.

The Boolean type.

Use the ST_GeometryFromText function to return two geometries. Then, use the ST_Touches function to check whether the two geometries have at least one point in common but their interiors do not intersect.

- Query statement

```
* |
SELECT
   ST_Touches(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    )
  )
```

- Query and analysis results

ST_Within function

The ST_Within function checks whether the first geometry is completely inside the second geometry. If yes, the function returns true.

```
ST_Within(,)
```

Parameter	Description
	The value of this parameter is of the geometry type.
	The value of this parameter is of the geometry type.

The Boolean type.

Use the ST_GeometryFromText function to return two geometries. Then, use the ST_Within function to check whether the first geometry is completely inside the second geometry.

- Query statement

```
* |
SELECT
  ST_Within(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    )
  )
```

- Query and analysis results

ST_Area function

The ST_Area function calculates the projected area of a geometry on a two-dimensional plane by using the Euclidean distance method.

```
ST_Area()
```

Parameter	Description
	The value of this parameter is of the geometry type.

The double type.

Use the ST_GeometryFromText function to return a geometry. Then, use the ST_Area function to calculate the projected area of the geometry on a two-dimensional plane.

- Query statement

```
* |
SELECT
  ST_Area(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- Query and analysis results

ST_Centroid function

The ST_Centroid function returns the point value that represents the mathematical centroid of a geometry.

```
ST_Centroid()
```

Parameter	Description
	The value of this parameter is of the geometry type.

The geometry type.

Use the ST_GeometryFromText function to return a geometry. Then, use the ST_Centroid function to return the point value that represents the mathematical centroid of the geometry.

- Query statement

```
* |
SELECT
  ST_Centroid(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- Query and analysis results

ST_CoordDim function

The ST_CoordDim function returns the coordinate dimension of a geometry.

```
ST_CoordDim()
```

Parameter	Description
	The value of this parameter is of the geometry type.

The bigint type.

Use the ST_GeometryFromText function to return a geometry. Then, use the ST_CoordDim function to return the coordinate dimension of the geometry.

- Query statement

```
* |
SELECT
  ST_CoordDim(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- Query and analysis results

_col0
2

ST_Dimension function

The ST_Dimension function returns the inherent dimension of a geometry. The inherent dimension must be less than or equal to the coordinate dimension.

```
ST_Dimension()
```

Parameter	Description
	The value of this parameter is of the geometry type. • If is a point or an empty geometry, the function returns 0. • If is a line string, the function returns 1. • If is a polygon, the function returns 2. • If is a geometry, the function returns the largest dimension of the collection.

The bigint type.

Use the ST_GeometryFromText function to return a geometry. Then, use the ST_Dimension function to return the inherent dimension of the geometry.

- Query statement

```
* |
SELECT
  ST_Dimension(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- Query and analysis results

ST_Distance function

The ST_Distance function returns the minimum distance between two geometries.

```
ST_Distance(,)
```

Parameter	Description
	The value of this parameter is of the geometry type.
	The value of this parameter is of the geometry type.

The double type.

Use the ST_GeometryFromText function to return two geometries.Then, use the ST_Distance function to return the minimum distance between the two geometries.

- Query statement

```
* |
SELECT
  ST_Distance(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 50)))'
    ),
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- Query and analysis results

ST_EndPoint function

The ST_EndPoint function returns the last point of a line string.

```
ST_EndPoint()
```

Parameter	Description
	The value of this parameter is of the geometry type.

The point type.

Use the ST_LineFromText function to return a line string. Then, use the ST_EndPoint function to return the last point of the line string.

- Query statement

```
* |
SELECT
  ST_EndPoint(
    ST_LineFromText(
      'linestring (10 10,20 20)'
    )
  )
```

- Query and analysis results

ST_IsClosed function

The ST_IsClosed function checks whether the start point of a line string coincides with the end point. If yes, the function returns true.

```
ST_IsClosed()
```

Parameter	Description
	The value of this parameter is of the geometry type.

The Boolean type.

Use the ST_LineFromText function to return a line string. Then, use the ST_IsClosed function to check whether the start point of the line string coincides with the end point.

- Query statement

```
* |
SELECT
  ST_IsClosed(
    ST_LineFromText(
      'linestring (10.05 10.28 , 20.95 20.89 )'
    )
  )
```

- Query and analysis results

ST_IsEmpty function

The ST_IsEmpty function checks whether a geometry is empty. If yes, the function returns true.

```
ST_IsEmpty()
```

Parameter	Description
	The value of this parameter is of the geometry type.

The Boolean type.

Use the ST_Point function to return a point. Then, use the ST_IsEmpty function to check whether the point is empty.

- Query statement

```
* | SELECT ST_IsEmpty(ST_Point(1,1))
```

- Query and analysis results

ST_IsRing function

The ST_IsRing function checks whether a line string is closed and simple. If yes, the function returns true.

```
ST_IsRing()
```

Parameter	Description
	The value of this parameter is of the geometry type.

The Boolean type.

Use the ST_LineFromText function to return a line string. Then, use the ST_IsRing function to check whether the line string is closed and simple.

- Query statement

```
* |
SELECT
  ST_IsRing(
    ST_LineFromText(
      'linestring (10.05 10.28,20.95 20.89 )'
    )
  )
```

- Query and analysis results

ST_Length function

The ST_Length function calculates the projected length of a line string on a two-dimensional plane by using the Euclidean distance method. If multiple line strings exist, the function returns the sum of the lengths of the multiple line strings.

```
ST_Length()
```

Parameter	Description
	The value of this parameter is of the geometry type.

The double type.

Use the ST_LineFromText function to return a line string. Then, use the ST_Length function to calculate the projected length of the line string.

- Query statement

```
* |
SELECT
  ST_Length(
    ST_LineFromText(
      'linestring (10.05 10.28,20.95 20.89)'
    )
  )
```

- Query and analysis results

ST_NumPoints function

The ST_NumPoints function returns the number of points in a geometry.

```
ST_NumPoints()
```

Parameter	Description
	The value of this parameter is of the geometry type.

The bigint type.

Use the ST_LineFromText function to return a line string. Then, use the ST_NumPoints function to return the number of points in the line string.

- Query statement

```
* |
SELECT
  ST_NumPoints(
    ST_LineFromText('linestring (10 10,20 20)')
  )
```

- Query and analysis results

ST_NumInteriorRing function

The ST_NumInteriorRing function returns the number of interior rings in a geometry.

```
ST_NumInteriorRing()
```

Parameter	Description
	The value of this parameter is of the geometry type.

The bigint type.

Use the ST_GeometryFromText function to return a geometry. Then, use the ST_NumInteriorRing function to return the number of interior rings in the geometry.

- Query statement

```
* |
SELECT
  ST_NumInteriorRing(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- Query and analysis results

_col0
1

ST_StartPoint function

The ST_StartPoint function returns the first point of a line string.

```
ST_StartPoint()
```

Parameter	Description
	The value of this parameter is of the geometry type.

The point type.

Use the ST_LineFromText function to return a line string. Then, use the ST_StartPoint function to return the first point of the line string.

- Query statement

```
* |
SELECT
  ST_StartPoint(
    ST_LineFromText(
      'linestring (10 10,20 20 )'
    )
  )
```

- Query and analysis results

_col0
POINT (10 10)

ST_X function

The ST_X function returns the X-coordinate of a specified point.

```
ST_X()
```

Parameter	Description
	The value of this parameter is of the point type.

The double type.

Use the ST_Point function to return a point. Then, use the ST_X function to return the X-coordinate of the point.

- Query statement

```
* | SELECT ST_X(ST_Point(1,3))
```

- Query and analysis results

_col0
1.0

ST_XMax function

The ST_XMax function returns the maximum first X-coordinate of a geometry.

```
ST_XMax()
```

Parameter	Description
	The value of this parameter is of the geometry type.

The double type.

Use the ST_GeometryFromText function to return a geometry. Then, use the ST_XMax function to return the maximum first X-coordinate of the geometry.

- Query statement

```
* |
SELECT
  ST_XMax(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- Query and analysis results

_col0
60.0

ST_XMin function

The ST_XMin function returns the minimum first X-coordinate of a geometry.

```
ST_XMin()
```

Parameter	Description
	The value of this parameter is of the geometry type.

The double type.

Use the ST_GeometryFromText function to return a geometry. Then, use the ST_XMin function to return the minimum first X-coordinate of the geometry.

- Query statement

```
* |
SELECT
  ST_XMin(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- Query and analysis results

ST_Y function

The ST_Y function returns the Y-coordinate of a specified point.

```
ST_Y()
```

Parameter	Description
	The value of this parameter is of the point type.

The double type.

Use the ST_Point function to return a point. Then, use the ST_Y function to return the Y-coordinate of the point.

- Query statement

```
* | SELECT ST_Y(ST_Point(1,3))
```

- Query and analysis results

ST_YMax function

The ST_YMax function returns the maximum first Y-coordinate of a geometry.

```
ST_YMax()
```

Parameter	Description
	The value of this parameter is of the geometry type.

The double type.

Use the ST_GeometryFromText function to return a geometry. Then, use the ST_YMax function to return the maximum first Y-coordinate of the geometry.

- Query statement

```
* |
SELECT
  ST_YMax(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- Query and analysis results

ST_YMin function

The ST_YMin function returns the minimum first Y-coordinate of a geometry.

```
ST_YMin()
```

Parameter	Description
	The value of this parameter is of the geometry type.

The double type.

Use the ST_GeometryFromText function to return a geometry. Then, use the ST_YMin function to return the minimum first Y-coordinate of the geometry.

- Query statement

```
* |
SELECT
  ST_YMin(
    ST_GeometryFromText(
      'multipolygon (((10 10,10 20,20 20,20 15,10 10), (50 40,50 50,60 50,60 40,50 40)))'
    )
  )
```

- Query and analysis results

bing_tile function

The bing_tile function returns a Bing tile.

- The following function returns a Bing tile based on the X-coordinate, Y-coordinate, and zoom level.

```
bing_tile(,, zoom_level)
```

- The following function returns a Bing tile based on the quadtree key.

```
bing_tile(quadKey)
```

Parameter	Description
	The X-coordinate. The value of this parameter is of the integer type.
	The Y-coordinate. The value of this parameter is of the integer type.
zoom_level	The zoom level. Valid values: [1,23]. The value of this parameter is of the integer type.
quadKey	The quadtree key.

The BingTile type.

- Example 1: Create a Bing tile based on the X-coordinate, Y-coordinate, and zoom level.
  - Query statement

    ```
    * | SELECT bing_tile(10, 20, 20)
    ```

  - Query and analysis results

- Example 2: Create a Bing tile based on the quadtree key.
  - Query statement

    ```
    * | SELECT bing_tile(bing_tile_quadkey(bing_tile(10, 20, 20)))
    ```

  - Query and analysis results

bing_tile_at function

The bing_tile_at function returns a Bing tile based on the latitude, longitude, and zoom level.

```
bing_tile_at(,,zoom_level)
```

Parameter	Description
	The latitude. Valid values: [-85.05112878,85.05112878]. The value of this parameter is of the double type.
	The longitude. Valid values: [-180,180]. The value of this parameter is of the double type.
zoom_level	The zoom level. Valid values: [1,23]. The value of this parameter is of the integer type.

The BingTile type.

Create a Bing tile.

- Query statement

```
* | SELECT bing_tile_at(47.265511, -122.465691, 12)
```

- Query and analysis results

bing_tile_coordinates function

The bing_tile_coordinates function returns the X- and Y-coordinates of a Bing tile.

```
bing_tile_coordinates()
```

Parameter	Description
	The value of this parameter is of the BingTile type.

The array(integer,integer) type.

Obtain the X- and Y-coordinates of a Bing tile.

- Query statement

```
* | SELECT bing_tile_coordinates(bing_tile_at(47.265511, -122.465691, 12))
```

- Query and analysis results

bing_tile_polygon function

The bing_tile_polygon function returns the polygon representation of a Bing tile.

```
bing_tile_polygon()
```

Parameter	Description
	The value of this parameter is of the BingTile type.

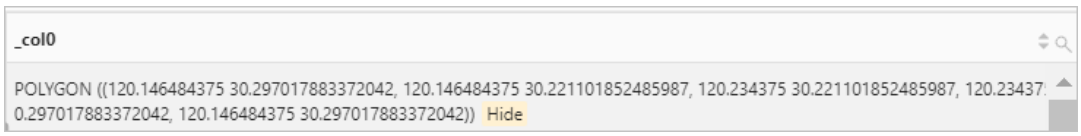
The polygon type.

Obtain the polygon representation of a Bing tile.

- Query statement

```
* | SELECT bing_tile_polygon(bing_tile_at(30.26, 120.19, 12))
```

- Query and analysis results



bing_tile_quadkey function

The bing_tile_quadkey function returns the quadtree key of a Bing tile.

```
bing_tile_quadkey()
```

Parameter	Description
	The value of this parameter is of the BingTile type.

The varchar type.

Obtain the quadtree key of a Bing tile.

- Query statement

```
* | SELECT bing_tile_quadkey(bing_tile(10, 20, 20))
```

- Query and analysis results

bing_tile_zoom_level function

The bing_tile_zoom_level function returns the zoom level of a Bing tile.

```
bing_tile_zoom_level()
```

Parameter	Description
	The value of this parameter is of the BingTile type.

The double type.

Obtain the zoom level of a Bing tile.

- Query statement

```
* | SELECT bing_tile_zoom_level(bing_tile(10, 20, 20))
```

- Query and analysis results

8.1.21. Geo functions

This topic describes the syntax of geo functions. This topic also provides examples on how to use the functions.

Notice

Function	Syntax	Description

Function	Syntax	Description
geohash function	geohash(*x*)	Encodes latitudes and longitudes by using the Geohash algorithm. **Note** Log Service allows you to convert IP addresses into geographic location information such as countries, provinces, cities, operators, and latitudes and longitudes. For more information, see IP functions.

geohash function

The geohash function encodes latitudes and longitudes by using the Geohash algorithm.

```
geohash(x)
```

Parameter	Description
	The value of this parameter is of the string type. The value indicates a latitude and a longitude. Example: `ip_to_geo(latitude,longitude)`.

The string type

Use the ip_to_geo function to convert the value of the client_ip field into a latitude and a longitude. Then, use the geohash function to encode the latitude and longitude.

- Query statement

```
* | SELECT geohash(ip_to_geo(client_ip)) AS hash
```

- Query and analysis results

8.1.22. Color functions

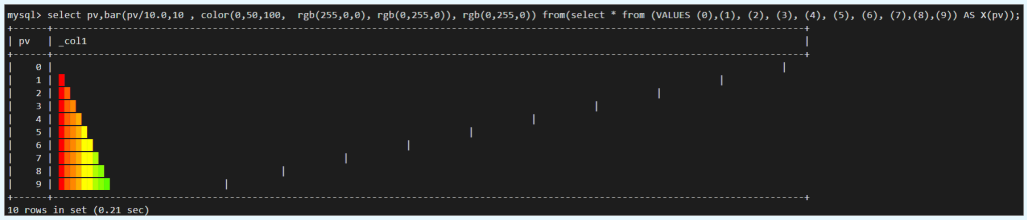
This topic describes the syntax of color functions. This topic also provides examples on how to use color functions.

Notice

- If you want to use strings in analytic statements, you must enclose the strings in single quotation marks (''). Strings that are not enclosed or are enclosed in double quotation marks ("") indicate field names or column names. For example, 'status' indicates the status string, and status or "status" indicates the status log field.
- When you use color functions in the Log Service console, the display of query and analysis results is compromised. To avoid this issue, we recommend that you view the query and analysis results on your server.
  - Display of query and analysis results in the console

  - Display of query and analysis results on a server

Function	Syntax	Description
bar function	bar(,*width*)	Returns a part of an ANSI bar chart. You can configure the *width* parameter to specify the width of the ANSI bar chart. However, you cannot configure the high_color or low_color parameter to specify the colors for the chart. The default values of the high_color and low_color parameters are used. The default value of the low_color parameter is red, and the default value of the high_color parameter is green. In addition, you can configure to specify the length of the part that is returned by the function.
	bar(,*width*,*low_color*,*high_color*)	Returns a part of an ANSI bar chart. You can configure the *width* parameter to specify the width of the ANSI bar chart. You can also configure the high_color and low_color parameters to specify custom colors for the chart. In addition, you can configure to specify the length of the part that is returned by the function.
color function	color(string)	Converts a color string to a color type.
	color(,*low*,*high*,*low_color*,*high_color*)	Returns a color between *high_color* and *low_color* based on the portions of *high_color* and *low_color*. The portions are determined by the proportion of between *high* and *low*.

Function	Syntax	Description
	color(,*low_color*, *high_color*)	Returns a color between *high_color* and *low_color* based on the portions of *high_color* and *low_color*. The portions are determined by .
render function	render(*boolean expression*)	Returns results by using color rendering. If the Boolean expression evaluates to true, the function returns a green tick. If the Boolean expression evaluates to false, the function returns a red cross.
	render(,*color*)	Returns results by using custom color rendering.
rgb function	rgb(*red*,*green*,*blue*)	Returns a color value based on an RGB value.

bar function

The bar function returns an ANSI bar chart.

- The following function returns a part of an ANSI bar chart. You can configure the *width* parameter to specify the width of the ANSI bar chart. However, you cannot configure the high_color or low_color parameter to specify the colors for the chart. The default values of the high_color and low_color parameters are used. The default value of the low_color parameter is red, and the default value of the high_color parameter is green. In addition, you can configure to specify the length of the part that is returned by the function.

```
bar(,width)
```

- The following function returns a part of an ANSI bar chart. You can configure the *width* parameter to specify the width of the ANSI bar chart. You can also configure the high_color and low_color parameters to specify custom colors for the chart. In addition, you can configure to specify the length of the part that is returned by the function.

```
bar(,width,low_color,high_color)
```

Parameter	Description
	The proportion of the part that is returned by the function to the ANSI bar chart. The value of this parameter is of the double type. Valid values: [0,1].
width	The width of the ANSI bar chart.
low_color	The RGB value of the start color.
high_color	The RGB value of the end color.

The varchar type.

- Example 1: Obtain a part of an ANSI bar chart based on the proportion of page views (PVs) within a specified hour to the total PVs.

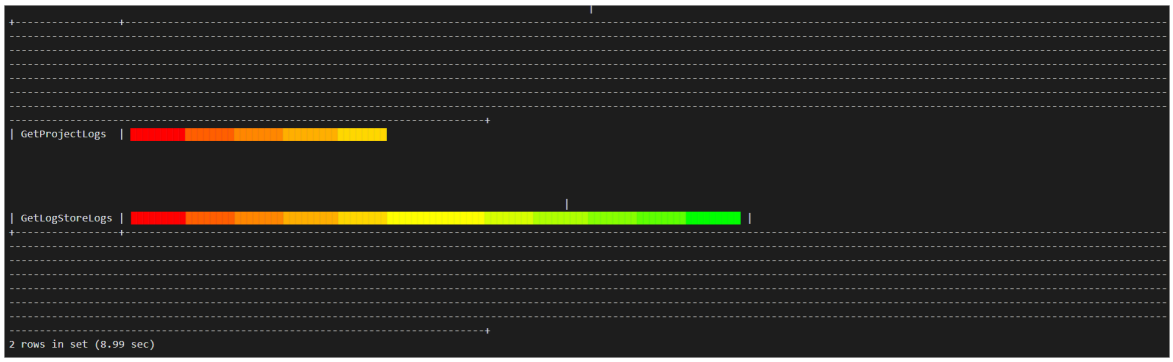
- Query statement

```
* |
SELECT
  Method,
  bar(pv/m,100)
FROM(
    SELECT
      *,
      max(pv) over() AS m
    FROM(
        SELECT
          Method,
          count(1) AS pv
        FROM         internal-operation_log
        WHERE
          __date__ > '2021-09-10 00:00:00'
          AND __date__ < '2021-09-10 01:00:00'
        GROUP BY
          Method
      )
  )
```

- Query and analysis results (console)

Method	_col1
GetProjectLogs	□[38;5;196m█□[38;5;196m█□[38;5;196m█□[38;5;196m █□[38;5;196m█□[38;5;196m█□[38;5;196m█□[38;5;196m █□[38... Show
GetLogStoreLogs	□[38;5;196m█□[38;5;202m█□[38;5;214m█□[38;5;220m █□[38;5;226m█□[38;5;190m█□[38;5;154m█□[38;5;82m █□[38;... Show

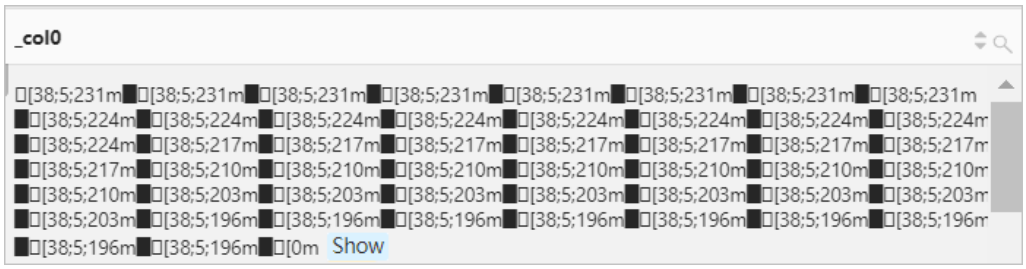
- Query and analysis results (server)



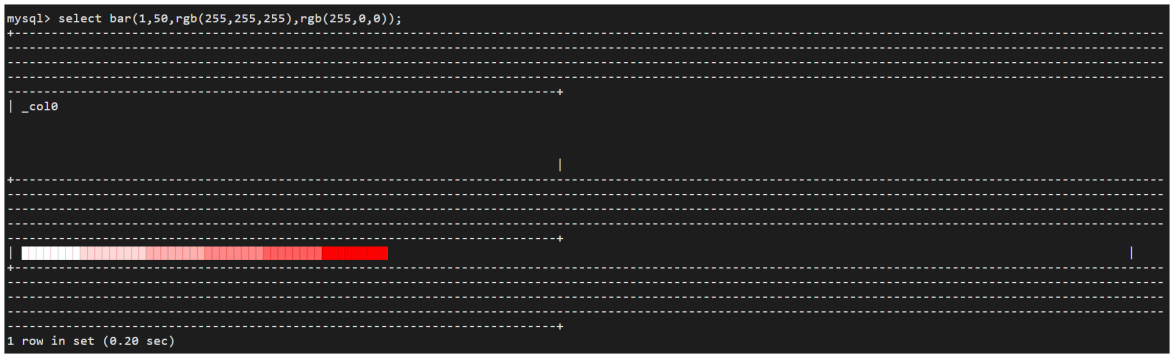
- Example 2: Obtain an ANSI bar chart that is displayed in red and white and has a width of 50.
  - Query statement

```
* | SELECT bar(1,50,rgb(255,255,255),rgb(255,0,0))
```

- Query and analysis results (console)



- Query and analysis results (server)



color function

The color function returns the color that corresponds to a value.

- The following function converts a color string to a color type:

```
color(string)
```

- The following function returns a color between *high_color* and *low_color* based on the portions of *high_color* and *low_color*. The portions are determined by the proportion of between *high* and *low*.

```
color(,low,high,low_color,high_color)
```

- The following function returns a color between *high_color* and *low_color* based on the portions of *high_color* and *low_color*. The portions are determined by .

```
color(,low_color,high_color)
```

Parameter	Description
	The value of this parameter is of the double type.
	The value of this parameter is of the double type. Valid values: [0,1].
low	The minimum value. The value of this parameter is of the double type.
high	The maximum value. The value of this parameter is of the double type.
low_color	The RGB value of the start color.
high_color	The RGB value of the end color.
string	The string. Valid values: black, red, green, yellow, blue, magenta, cyan, and white. The value can also be an RGB value in the Cascading Style Sheet (CSS) format. Example: #000.

The color type.

- Example 1: Convert a color string to a color type.
  - Query statement

```
* | SELECT color('#000')
```

  - Query and analysis results (console)

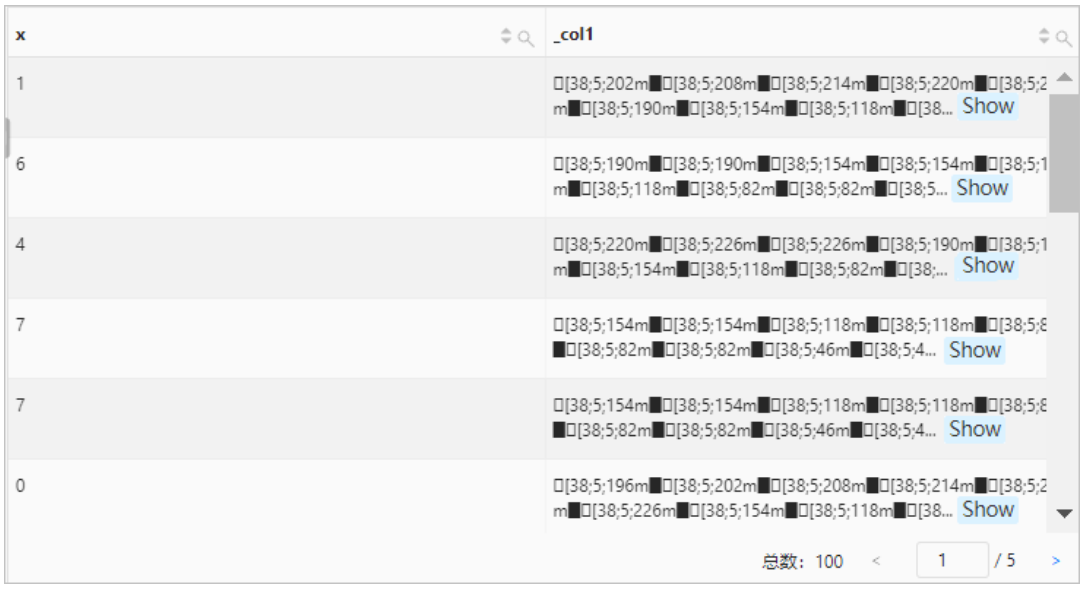
  - Query and analysis results (server)

```
mysql> SELECT color('#000');
+---------+
| _col0   |
+---------+
| #000000 |
+---------+
1 row in set (0.21 sec)
```

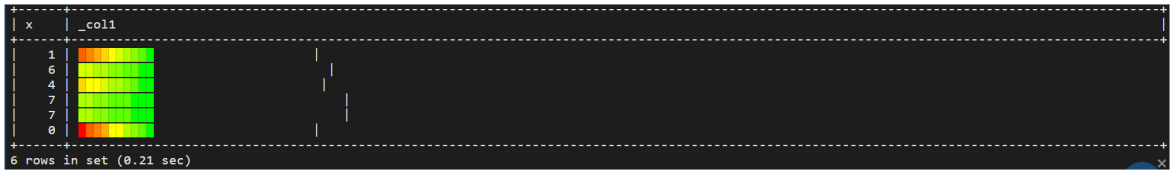
- Example 2: Obtain a part of an ANSI bar chart. The remainder of the request_length field value is calculated. Then, the color function returns a color that corresponds to the remainder, and the bar function returns a part of an ANSI bar chart based on the color.
  - Query statement

```
*|SELECT x,bar(10,10, color(x, 0,10, rgb(255,0,0), rgb(0,255,0)), rgb(0,255,0)) FROM(SELECT  *FR
OM (SELECT  request_length%10 x FROM  log))
```

  - Query and analysis results (console)



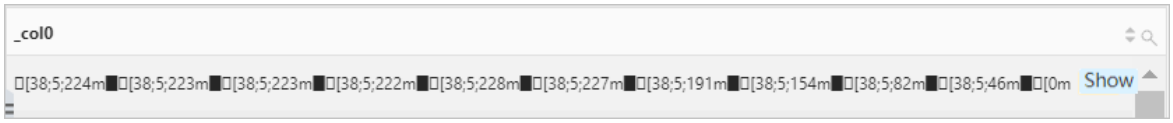
  - Query and analysis results (server)



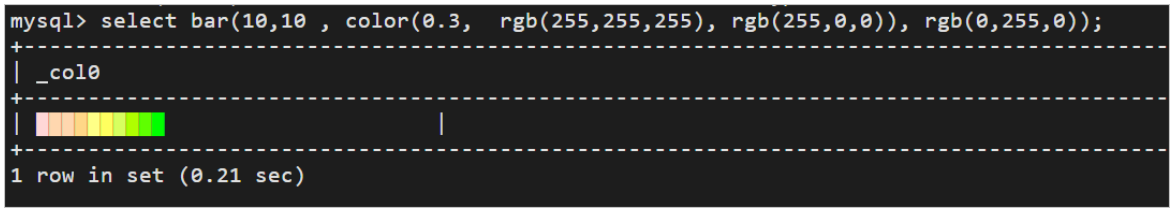
- Example 3: Obtain a part of an ANSI bar chart. The color function returns a color, and the bar function returns a part of an ANSI bar chart based on the color.
  - Query statement

```
*|SELECT bar(10,10, color(0.3, rgb(255,255,255), rgb(255,0,0)), rgb(0,255,0))
```

  - Query and analysis results (console)



  - Query and analysis results (server)



render function

The render function returns results by using color rendering.

- The following function returns results by using color rendering: If the Boolean expression evaluates to true, the function returns a green tick. If the Boolean expression evaluates to false, the function returns a red cross.

```
render(boolean expression)
```

- The following function returns results by using custom color rendering:

```
render(,color)
```

Parameter	Description
boolean expression	The Boolean expression.
	The X coordinate. The value of this parameter is of the integer type.
color	The color. The value of this parameter is of the color type.

The varchar type.

- Example 1: Check whether the number of PVs is less than 1,000. The count function returns the number of PVs, and the render function determines whether the number of PVs is less than 1,000 and returns results by using color rendering. If the number of PVs is less than 1,000, the render function returns a green tick.
  - Query statement

```
* | SELECT render(count(*)<1000)
```

  - Query and analysis results (console)

- Query and analysis results (server)

```
mysql> select render(count(*)<1000);
+------------------+
| _col0            |
+------------------+
| ✓   |
+------------------+
1 row in set (0.22 sec)
```

- Example 2: Obtain the total number of logs by using green rendering. The count function returns the total number of logs, and the render function returns results by using green rendering.
  - Query statement

```
* | SELECT render(count(*),rgb(48,169,16))
```

  - Query and analysis results (console)

  - Query and analysis results (server)

```
mysql> select render(count(*),rgb(48,169,16));
+------------------+
| _col0            |
+------------------+
| 1 |
+------------------+
1 row in set (0.21 sec)
```

rgb function

The rgb function returns a color value based on an RGB value.

```
rgb(red,green,blue)
```

Parameter	Description
red	The portion of red. Valid values: [0,255]. The value of this parameter is of the integer type.
green	The portion of green. Valid values: [0,255]. The value of this parameter is of the integer type.
blue	The portion of blue. Valid values: [0,255]. The value of this parameter is of the integer type.

The color type.

Obtain a color value based on an RGB value.

- Query statement

```
*|SELECT rgb(255,0,0)
```

- Query and analysis results (console)

- Query and analysis results (server)

```
mysql> select rgb(255,0,0);
+---------+
| _col0   |
+---------+
| #ff0000 |
+---------+
1 row in set (0.21 sec)
```

8.1.23. HyperLogLog functions

HyperLogLog functions are approximate aggregate functions and are similar to the approx_distinct function. If a large amount of data is involved in computation, HyperLogLog functions can be used to return estimation results within a shorter period of time. This topic describes the syntax of HyperLogLog functions. This topic also provides examples on how to use HyperLogLog functions.

Notice

Function	Syntax	Description
approx_set function	approx_set()	Estimates the number of distinct values in the field. The maximum standard error is 0.01625, which is the default value.
cardinality function	cardinality()	Converts HyperLogLog data to bigint data.
empty_approx_set function	empty_approx_set()	Returns a null value of the HyperLogLog type. The maximum standard error is 0.01625, which is the default value.
merge function	merge()	Aggregates all HyperLogLog values.

approx_set function

The approx_set function estimates the number of distinct values in the field. The maximum standard error is 0.01625, which is the default value.

```
approx_set()
```

Parameter	Description
	The value of this parameter is of an arbitrary data type.

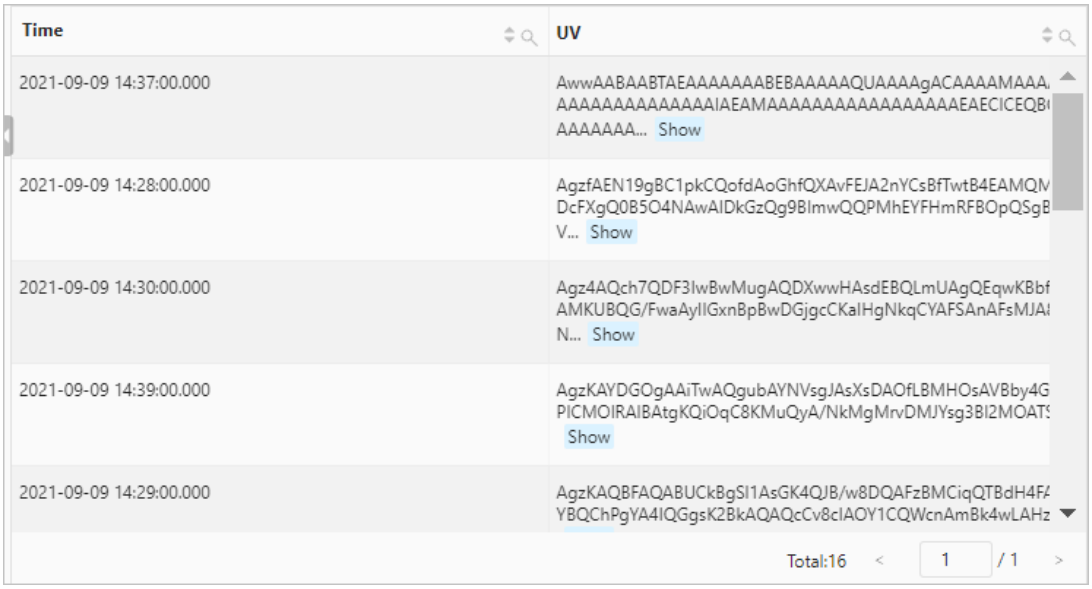
The HyperLogLog type.

Estimate the number of unique visitors (UVs) per minute. The return value is of the HyperLogLog type.

- Query statement

```
* |
SELECT
  date_trunc('minute', __time__) AS Time,
  approx_set(client_ip) AS UV
FROM  website_log
GROUP BY
  Time
ORDER BY
  Time
```

- Query and analysis results



cardinality function

The cardinality function converts HyperLogLog data to bigint data.

```
cardinality()
```

Parameter	Description
	The value of this parameter is of the HyperLogLog type.

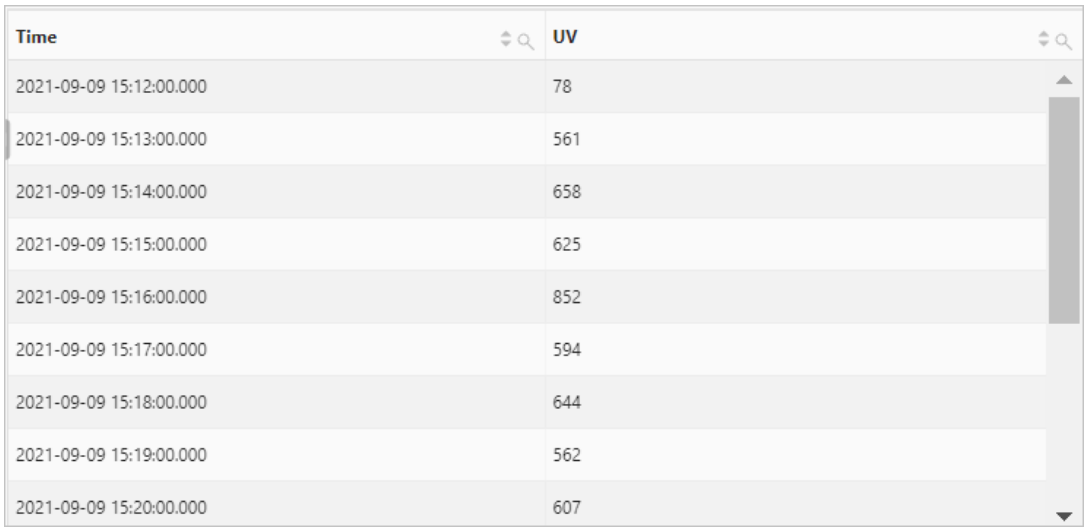
The bigint type.

Convert HyperLogLog data to bigint data. The approx_set function returns the estimated number of UVs per minute. The return value is of the HyperLogLog type. The cardinality function converts the return value to bigint data.

- Query statement

```
* |
SELECT
  Time,
  cardinality(UV) AS UV
FROM  (
    SELECT
      date_trunc('minute', __time__) AS Time,
      approx_set(client_ip) AS UV
    FROM      website_log
    GROUP BY
      Time
    ORDER BY
      Time
  ) AS UV
```

- Query and analysis results



Time	UV
2021-09-09 15:12:00.000	78
2021-09-09 15:13:00.000	561
2021-09-09 15:14:00.000	658
2021-09-09 15:15:00.000	625
2021-09-09 15:16:00.000	852
2021-09-09 15:17:00.000	594
2021-09-09 15:18:00.000	644
2021-09-09 15:19:00.000	562
2021-09-09 15:20:00.000	607

empty_approx_set function

The empty_approx_set function returns a null value of the HyperLogLog type. The maximum standard error is 0.01625, which is the default value.

```
empty_approx_set()
```

The HyperLogLog type.

Obtain a null value of the HyperLogLog type.

- Query statement

```
* | SELECT  empty_approx_set()
```

- Query and analysis results

_col0
AgwAAA==

merge function

The merge function aggregates all HyperLogLog values.

```
merge()
```

Parameter	Description
	The value of this parameter is of the HyperLogLog type.

The HyperLogLog type.

Aggregate HyperLogLog values. The approx_set function returns the estimated number of UVs per minute. The merge function aggregates the numbers of UVs of 15 minutes. The cardinality function converts the HyperLogLog data into bigint data.

- Query statement

```
* |
SELECT
  Time,
  cardinality(UV) AS UV,
  cardinality(merge(UV) over()) AS Total_UV
FROM  (
    SELECT
      date_trunc('minute', __time__) AS Time,
      approx_set(client_ip) AS UV
    FROM      log
    GROUP BY
      Time
    ORDER BY
      Time
  )
```

- Query and analysis results

Time	UV	Total_UV
2021-09-09 15:13:00.000	561	8564
2021-09-09 15:14:00.000	658	8564
2021-09-09 15:15:00.000	625	8564
2021-09-09 15:16:00.000	852	8564
2021-09-09 15:17:00.000	594	8564
2021-09-09 15:18:00.000	644	8564
2021-09-09 15:19:00.000	562	8564
2021-09-09 15:20:00.000	607	8564
2021-09-09 15:21:00.000	611	8564

总数：15 < 1 / 1 >

8.1.24. Comparison operators

Comparison operators are used to compare parameter values. The values that are of the following data types can be compared: double, bigint, varchar, timestamp, and date. This topic describes the syntax of comparison operators. This topic also provides examples on how to use the comparison operators.

Notice

Operator	Syntax	Description
Relational operators	<	If is less than , true is returned.
	>	If is greater than , true is returned.
	<=	If is less than or equal to , true is returned.
	>=	If is greater than or equal to , true is returned.
	=	If is equal to , true is returned.
	<>	If is not equal to , true is returned.
	!=	If is not equal to , true is returned.
all operator	*<Relational operator>* all(*subquery*)	If meets all conditions, true is returned.
any operator	*<Relational operator>* any(*subquery*)	If meets one of the conditions, true is returned.
between operator	between and	If is between and , true is returned.
distinct operator	is distinct from	If is not equal to , true is returned.
	is not distinct from	If is equal to , true is returned.
like operator	like *pattern* [escape '*escape_character*']	Is used to match a specified character pattern in a string. The string is case-sensitive.
some operator	*<Relational operator>* some(*subquery*)	If meets one of the conditions, true is returned.
greatest operator	greatest(,...)	Is used to obtain the greater value of and .
least operator	least(,...)	Is used to obtain the smaller value of and .
null operator	is null	If is null, true is returned.
	is not null	If is not null, true is returned.

Relational operators

Relational operators are used to compare and . If the condition is met, true is returned.

Syntax	Description
<	is less than .
>	is greater than .
<=	is less than or equal to .
>=	is greater than or equal to .

Syntax	Description
=	is equal to .
<>	is not equal to .
!=	is not equal to .

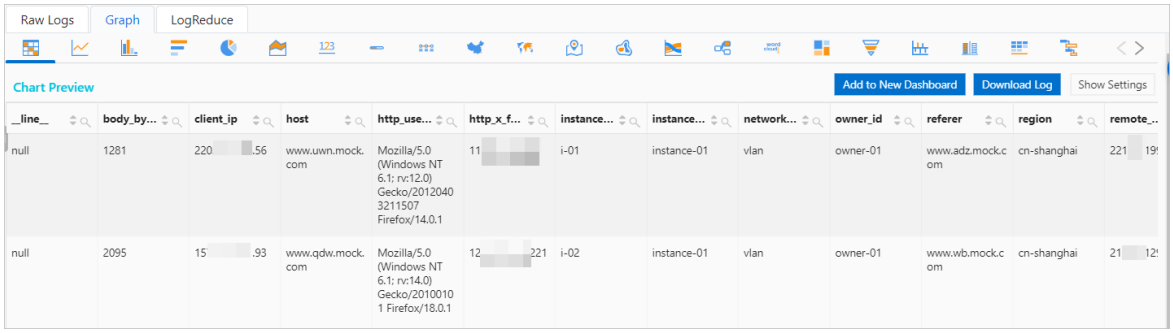
Parameter	Description
	The value of this parameter is of a data type that supports comparison.
	The value of this parameter is of a data type that supports comparison.

The Boolean type.

- Example 1: Query logs from the previous day.
  - Query statement

```
* |
SELECT
  *
FROM  log
WHERE
  __time__ < to_unixtime(current_date)
  AND __time__ > to_unixtime(date_add('day', -1, current_date))
```

  - Query and analysis results



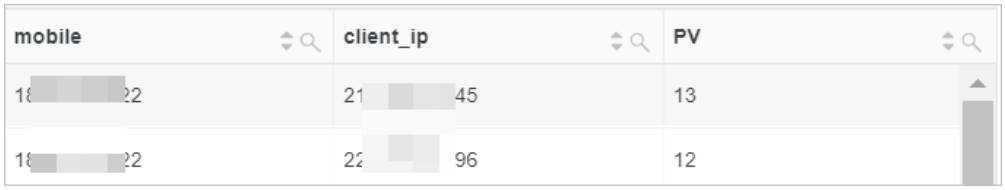
- Example 2: E-commerce Company A uses the mobile and client_ip fields in access logs to find the customers whose phone numbers are from a different place than the IP addresses of the accessed websites.
  - Sample field

```
mobile:1881111****
client_ip:192.168.2.0
```

- Query statement

```
* |
SELECT
  mobile,
  client_ip,
  count(*) as PV
WHERE
  mobile_city(mobile) != ip_to_city(client_ip)
  AND ip_to_city(client_ip) != ''
GROUP BY
  client_ip,
  mobile
ORDER BY
  PV DESC
```

- Query and analysis results



all operator

The all operator is used to determine whether meets all conditions. If all conditions are met, true is returned.

```
<Relational operator> all(subquery)
```

Parameter	Description
	The value of this parameter is of a data type that supports comparison.
Relational operator	The value of this parameter is a relational operator. Valid values: < > <= >= = <> != **Notice** The all operator must follow the relational operator. Relational operators: < > <= >= = <> !=
subquery	The value of this parameter is an SQL subquery.

The Boolean type.

Check whether each request to instance i-01 is responded with status code 200.

- Sample field

```
instance_id:i-01
status:200
```

- Query statement

```
* | select 200 = all(select status where instance_id='i-01')
```

- Query and analysis results

any operator

The any operator is used to determine whether meets one of the conditions. If one of the conditions is met, true is returned.

```
<Relational operator> any(subquery)
```

Parameter	Description
	The value of this parameter is of a data type that supports comparison.
Relational operator	The value of this parameter is a relational operator. Valid values: < > <= >= = <> != **Notice** The any operator must follow the relational operator. Relational operators: < > <= >= = <> !=
subquery	The value of this parameter is an SQL subquery.

The Boolean type.

Check whether any request to instance i-01 is responded with status code 200.

- Sample field

```
instance_id:i-01
status:200
```

- Query statement

```
* | select 200 = any(select status where instance_id='i-01')
```

- Query and analysis results

between operator

The between operator is used to determine whether is between and . If the condition is met, true is returned. and specify a closed interval.

```
between  and
```

Parameter	Description
	The value of this parameter is of a data type that supports comparison.
	The value of this parameter is of a data type that supports comparison.
	The value of this parameter is of a data type that supports comparison.

> **Notice**
> - The data types of , , and must be the same.
> - If the value of , , or contains null, null is returned.

The Boolean type.

- Example 1: Determine whether the value of the status field is within the [200,299] range.
  - Query statement

    ```
    * | select status between 200 and 299
    ```

  - Query and analysis results

- Example 2: Determine the number of logs whose value of the status field is not within the [200,299] range.
  - Query statement

    ```
    * | select count(*) as count from log where status not between 200 and 299
    ```

  - Query and analysis results

distinct operator

The distinct operator is used to determine whether is equal to .

- is distinct from: If is not equal to , true is returned.

  ```
  is distinct from
  ```

- is not distinct from: If is equal to , true is returned.

  ```
  is not distinct from
  ```

Parameter	Description
	The value of this parameter is of a data type that supports comparison.
	The value of this parameter is of a data type that supports comparison.

Unlike the = and <> operators, the distinct operator can be used to perform comparison on null.

		=	<>	is distinct from	is not distinct from
1	1	true	false	false	true

		=	<>	is distinct from	is not distinct from
1	2	false	true	true	false
1	null	null	null	true	false
null	null	null	null	false	true

The Boolean type.

Compare 0 against null.

- Query statement

```
* | select 0 is distinct from null
```

- Query and analysis results

like operator

The like operator is used to match a specified character pattern in a string. The string is case-sensitive.

```
like pattern [escape 'escape_character']
```

Parameter	Description
	The value of this parameter is of a data type that supports comparison.
pattern	The value of this parameter is the character pattern, which can contain strings or wildcards. The following wildcards are supported: • The percent sign (%) indicates an arbitrary number of characters. • The underscore (_) indicates a single character.
escape_character	The value of this parameter is a character expression that is used to escape the wildcard characters in the character pattern.

Note The like operator is used to query logs based on exact match. For more information, see How do I query logs by using exact match?

The Boolean type.

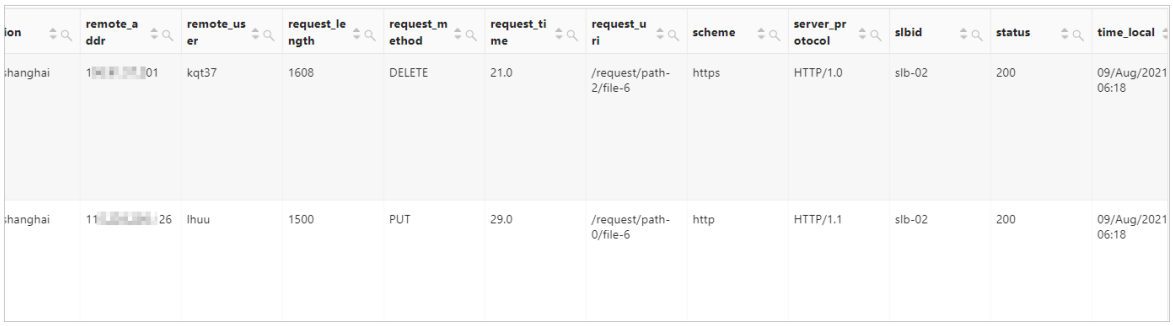
- Example 1: Search for the logs whose value of the request_uri field ends with file-8 or file-6.
  - Sample field

```
request_uri:/request/path-2/file-6
```

  - Query statement

```
*|select *  where request_uri like '%file-8' OR request_uri like '%file-6'
```

- Query and analysis results



ion	remote_addr	remote_user	request_length	request_method	request_time	request_uri	scheme	server_protocol	slbid	status	time_local
shanghai	1[redacted]01	kqt37	1608	DELETE	21.0	/request/path-2/file-6	https	HTTP/1.0	slb-02	200	09/Aug/2021 06:18
shanghai	11[redacted]26	lhuu	1500	PUT	29.0	/request/path-0/file-6	http	HTTP/1.1	slb-02	200	09/Aug/2021 06:18

- Example 2: Check whether the value of the request_uri field ends with file-6.
  - Sample field

```
request_uri:/request/path-2/file-6
```

  - Query statement

```
* | select request_uri like '%file-6'
```

  - Query and analysis results

_col0
true

some operator

The some operator is used to determine whether meets one of the conditions. If one of the conditions is met, true is returned.

```
<Relational operator> some(subquery)
```

Parameter	Description
	The value of this parameter is of a data type that supports comparison.
Relational operator	The value of this parameter is a relational operator. Valid values: < > <= >= = <> != **Notice** The some operator must follow the relational operator. Relational operators: < > <= >= = <> !=
subquery	The value of this parameter is an SQL subquery.

The Boolean type.

Check whether any request to instance i-01 is processed for less than 20s.

- Sample field

```
instance_id:i-01
request_time:16
```

- Query statement

```
* | select 20 > some(select request_time where instance_id='i-01')
```

- Query and analysis results

greatest operator

The greatest operator is used to obtain the greater value of and .

> **Note** The greatest operator is used for horizontal comparison, and the max function is used for vertical comparison.

```
greatest(,...)
```

Parameter	Description
	The value of this parameter is of a data type that supports comparison.
	The value of this parameter is of a data type that supports comparison.

The double type.

Compare the values of the request_time and status fields in the same line to obtain the greater value.

- Sample field

```
request_time:38
status:200
```

- Query statement

```
* |  SELECT greatest(request_time,status)
```

- Query and analysis results

least operator

The least operator is used to obtain the smaller value of and .

> **Note** The least operator is used for horizontal comparison, and the min function is used for vertical comparison.

```
least(,...)
```

Parameter	Description
	The value of this parameter is of a data type that supports comparison.
	The value of this parameter is of a data type that supports comparison.

The double type.

Compare the values of the request_time and status fields in the same line to obtain the smaller value.

- Sample field

```
request_time:77
status:200
```

- Query statement

```
* |  SELECT least(request_time,status)
```

- Query and analysis results

null operator

The null operator is used to determine whether is null.

- is null: If the parameter value is null, true is returned.

```
 is null
```

- is not null: If the parameter value is not null, true is returned.

```
 is not null
```

Parameter	Description
	The value of this parameter is of a data type that supports comparison.

The Boolean type.

- Example1: Determine whether the value of the status field is null.
  - Query statement

```
* | select status is null
```

  - Query and analysis results

- Example 2: Determine the number of logs whose status field is not empty.
  - Query statement

```
* | select count(*) as count from log where status is not null
```

  - Query and analysis results

8.1.25. Logical operators

This topic describes the syntax of logical operators. This topic also provides examples on how to use the logical operators.

Notice

- If you want to use strings in analytic statements, you must enclose the strings in single quotation marks (''). Strings that are not enclosed or are enclosed in double quotation marks ("") are considered field names or column names. For example, 'status' is considered the status string, and status or "status" is considered a log field whose name is status.
- The following logical operators are in descending order of priority: not, and, or. You can use parentheses () to change the calculation order.
- Logical operations support only Boolean expressions whose input value is true, false, or null.

Operator	Syntax	Description
AND operator	AND	If both and evaluate to true, the result is true.
OR operator	OR	If either or evaluates to true, the result is true.
NOT operator	NOT	If evaluates to false, the result is true.

AND operator

If both and evaluate to true, the result is true.

```
AND
```

Parameter	Description
	The value of this parameter is a Boolean expression.
	The value of this parameter is a Boolean expression.

The Boolean type.

If the value of the status field is 200 and the value of the request_method field is GET, true is returned. Otherwise, false is returned.

- Query statement

```
*|select status=200 AND request_method='GET'
```

- Query and analysis results

_col0
true
false

OR operator

If either or evaluates to true, the result is true.

```
OR
```

Parameter	Description
	The value of this parameter is a Boolean expression.
	The value of this parameter is a Boolean expression.

The Boolean type.

Search for the logs whose value of the request_uri field ends with file-8 or file-6.

- Query statement

```
*|SELECT *  WHERE request_uri LIKE '%file-8' OR request_uri LIKE '%file-6'
```

- Query and analysis results

ion	remote_addr	remote_user	request_length	request_method	request_time	request_uri	scheme	server_protocol	slbid	status	time_local
shanghai	1[redacted]01	kqt37	1608	DELETE	21.0	/request/path-2/file-6	https	HTTP/1.0	slb-02	200	09/Aug/2021 06:18
shanghai	11[redacted]26	lhuu	1500	PUT	29.0	/request/path-0/file-6	http	HTTP/1.1	slb-02	200	09/Aug/2021 06:18

NOT operator

If evaluates to false, the result is true.

```
NOT
```

Parameter	Description
	The value of this parameter is a Boolean expression.

The Boolean type.

Measure the durations of requests for which the HTTP 200 status code is not returned.

- Query statement

```
*|SELECT request_time WHERE NOT status=200
```

- Query and analysis results

request_time
53.0
24.0
56.0
32.0

Additional information: Truth table

The following table describes the results if and evaluate to true, false, or null.

		AND	OR	NOT
true	true	true	true	false
true	false	false	true	false
true	null	null	true	false
false	true	false	true	true
false	false	false	false	true
false	null	false	null	true
null	true	null	true	null
null	false	false	null	null
null	null	null	null	null

8.1.26. Unit conversion functions

Log Service allows you to convert a measurement or a time interval from the current unit to a different unit by using unit conversion functions. This topic describes the syntax of unit conversion functions. This topic also provides examples on how to use unit conversion functions.

Notice

Type	Function	Syntax	Description
Unit conversion for	convert_data_size function	convert_data_size()	Converts a measurement from the current unit to the optimal unit. The system automatically determines the optimal unit and returns a measurement in the optimal unit. The returned result is of the string type. For example, you can convert 1,024 KB to 1 MB and 1,024 MB to 1 GB.
		convert_data_size(, *unit*)	Converts a measurement from the current unit to a specified unit. The returned result is of the string type.
	format_data_size function	format_data_size(, *unit*)	Converts a measurement in byte to a measurement in a specified unit. The returned result is of the string type.
	parse_data_size function	parse_data_size()	Converts a measurement from the current unit to a measurement in byte. The returned result is of the decimal type.
	to_data_size_B function	to_data_size_B()	Converts a measurement from the current unit to a measurement in byte. The returned result is of the double type.

measurements Type	Function	Syntax	Description
	to_data_size_KB function	to_data_size_KB()	Converts a measurement from the current unit to a measurement in KB. The returned result is of the double type.
	to_data_size_MB function	to_data_size_MB()	Converts a measurement from the current unit to a measurement in MB. The returned result is of the double type.
	to_data_size_GB function	to_data_size_GB()	Converts a measurement from the current unit to a measurement in GB. The returned result is of the double type.
	to_data_size_TB function	to_data_size_TB()	Converts a measurement from the current unit to a measurement in TB. The returned result is of the double type.
	to_data_size_PB function	to_data_size_PB()	Converts a measurement from the current unit to a measurement in PB. The returned result is of the double type.
Unit conversion for time intervals	format_duration function	format_duration()	Converts a time interval in seconds to a readable string.
	parse_duration function	parse_duration()	Converts a time interval to a time interval in the `0 00:00:00.000` format.
	to_days function	to_days()	Converts a time interval to a time interval in days.
	to_hours function	to_hours()	Converts a time interval to a time interval in hours.
	to_microseconds function	to_microseconds()	Converts a time interval to a time interval in microseconds.
	to_milliseconds function	to_milliseconds()	Converts a time interval to a time interval in milliseconds.
	to_minutes function	to_minutes()	Converts a time interval to a time interval in minutes.
	to_most_succinct_time_unit function	to_most_succinct_time_unit()	Converts a time interval from the current unit to the optimal unit. The system automatically determines the optimal unit and returns a time interval in the optimal unit.
	to_nanoseconds function	to_nanoseconds()	Converts a time interval to a time interval in nanoseconds.
	to_seconds function	to_seconds()	Converts a time interval to a time interval in seconds.

convert_data_size function

The convert_data_size function converts a measurement from the current unit to a different unit.

- The following function converts a measurement from the current unit to the optimal unit. The system automatically determines the optimal unit and returns a measurement in the optimal unit.

```
convert_data_size()
```

- The following function converts a measurement from the current unit to a specified unit.

```
convert_data_size(, unit)
```

Parameter	Description
	The measurement. The value of this parameter is of the string type.
unit	The unit of stored data. Valid values: KB, MB, GB, PB, TB, EB, ZB, and YB.

The string type.

- Example 1: Convert 1,200 KB to a measurement in a different unit.
  - Query statement

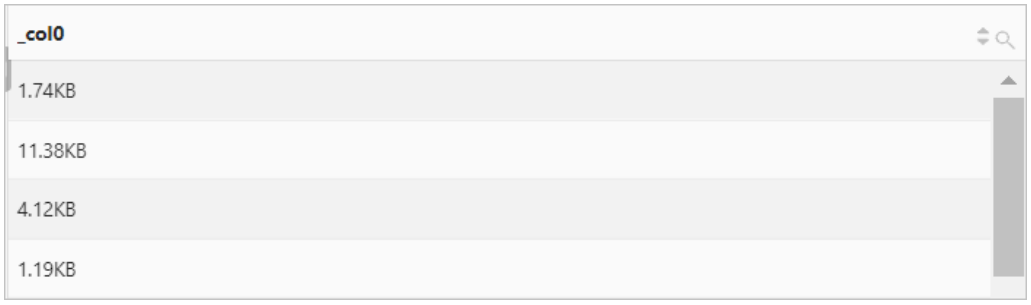
```
* | SELECT convert_data_size('1200KB')
```

  - Query and analysis results

- Example 2: Convert the value of the body_bytes_sent field in byte to a measurement in KB. The body_bytes_sent field indicates the number of bytes that are sent to the client.
  - Query statement

```
* | select convert_data_size(format_data_size(body_bytes_sent, 'KB'))
```

  - Query and analysis results



format_data_size function

The format_data_size function converts a measurement in byte to a measurement in a specified unit.

```
format_data_size(, unit)
```

Parameter	Description
	The measurement in byte. The value of this parameter is of the bigint type.

Parameter	Description
unit	The unit of stored data. Valid values: KB, MB, GB, PB, TB, EB, ZB, and YB.

The string type.

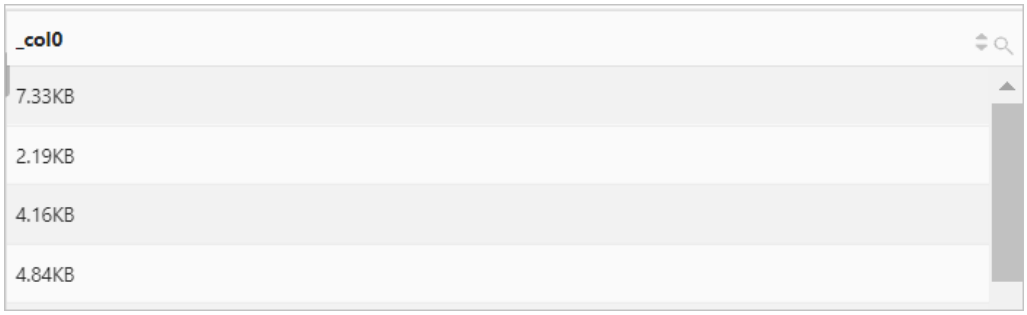
- Example 1: Convert the value of the body_bytes_sent field in byte to a measurement in KB. The body_bytes_sent field indicates the number of bytes that are sent to the client.
  - Sample field

    ```
    body_bytes_sent:4619
    ```

  - Query statement

    ```
    * | select format_data_size(body_bytes_sent, 'KB')
    ```

  - Query and analysis results

- Example 2: Convert the total number of bytes to a measurement in GB. The total number of bytes is calculated by adding up all the values of the body_bytes_sent field by using the sum function. The body_bytes_sent field indicates the number of bytes that are sent to the client.
  - Sample field

    ```
    body_bytes_sent:4619
    ```

  - Query statement

    ```
    * | select format_data_size(sum(body_bytes_sent), 'GB')
    ```

  - Query and analysis results

parse_data_size function

The parse_data_size function converts the current measurement to a measurement in byte.

```
parse_data_size()
```

Parameter	Description
	The measurement. The value of this parameter is of the string type.

The decimal type.

Convert 1,024 KB to a measurement in byte.

- Query statement

```
*| SELECT parse_data_size('1024KB')
```

- Query and analysis results

to_data_size_B function

The to_data_size_B function converts the current measurement to a measurement in byte.

```
to_data_size_B()
```

Parameter	Description
	The measurement. The value of this parameter is of the string type.

The double type.

Convert 1,024 KB to a measurement in byte.

- Query statement

```
* | select to_data_size_B('1024KB')
```

- Query and analysis results

to_data_size_KB function

The to_data_size_KB function converts the current measurement to a measurement in KB.

```
to_data_size_KB()
```

Parameter	Description
	The measurement. The value of this parameter is of the string type.

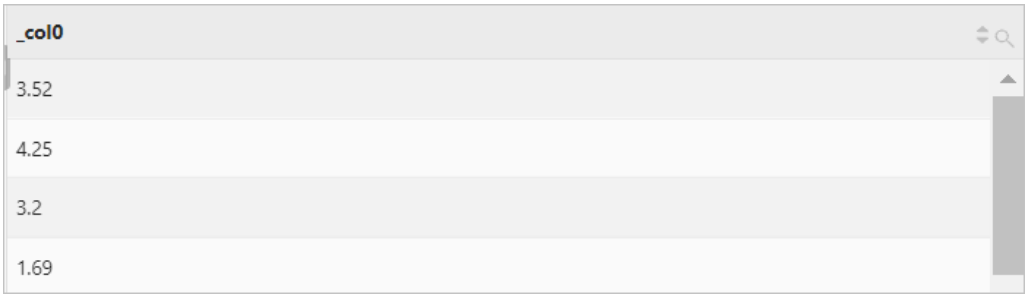
The double type.

Convert the value of the body_bytes_sent field to a measurement in KB. The body_bytes_sent field indicates the number of bytes that are sent to the client.

- Query statement

```
* | select to_data_size_KB(format_data_size(body_bytes_sent, 'KB'))
```

- Query and analysis results



to_data_size_MB function

The to_data_size_MB function converts the current measurement to a measurement in MB.

```
to_data_size_MB()
```

Parameter	Description
	The measurement. The value of this parameter is of the string type.

The double type.

Convert the total number of bytes to a measurement in MB. The total number of bytes is calculated by adding up all the values of the body_bytes_sent field by using the sum function. The body_bytes_sent field indicates the number of bytes that are sent to the client.

- Query statement

```
* | select to_data_size_MB(format_data_size(sum(body_bytes_sent), 'KB'))
```

- Query and analysis results

to_data_size_GB function

The to_data_size_GB function converts the current measurement to a measurement in GB.

```
to_data_size_GB()
```

Parameter	Description
	The measurement. The value of this parameter is of the string type.

The double type.

Convert the total number of bytes to a measurement in GB. The total number of bytes is calculated by adding up all the values of the body_bytes_sent field by using the sum function. The body_bytes_sent field indicates the number of bytes that are sent to the client.

- Query statement

```
* | select to_data_size_GB(format_data_size(sum(body_bytes_sent), 'KB'))
```

- Query and analysis results

to_data_size_TB function

The to_data_size_TB function converts the current measurement to a measurement in TB.

```
to_data_size_TB()
```

Parameter	Description
	The measurement. The value of this parameter is of the string type.

The double type.

Convert the total number of bytes to a measurement in TB. The total number of bytes is calculated by adding up all the values of the body_bytes_sent field by using the sum function. The body_bytes_sent field indicates the number of bytes that are sent to the client.

- Query statement

```
* | select to_data_size_TB(format_data_size(sum(body_bytes_sent), 'KB'))
```

- Query and analysis results

to_data_size_PB function

The to_data_size_PB function converts the current measurement to a measurement in PB.

```
to_data_size_PB()
```

Parameter	Description
	The measurement. The value of this parameter is of the string type.

The double type.

Convert 1,048,576 GB to a measurement in PB.

- Query statement

```
*| SELECT to_data_size_PB('1048576GB')
```

- Query and analysis results

format_duration function

The format_duration function converts a time interval in seconds to a readable string.

```
format_duration()
```

Parameter	Description
	The time interval. The value of this parameter is of the double type.

The string type.

Convert 235 seconds to a string in the `3 minutes, 55 seconds` format.

- Query statement

```
* | SELECT format_duration(235)
```

- Query and analysis results

parse_duration function

The parse_duration function converts a time interval to a time interval in the `0 00:00:00.000` format.

```
parse_duration()
```

Parameter	Description
	The time interval. The value of this parameter is of the string type.

The interval type.

Convert 1,340 milliseconds to a time interval in the `0 00:00:01.340` format.

- Query statement

```
* | SELECT parse_duration('1340ms')
```

- Query and analysis results

to_days function

The to_days function converts a time interval to a time interval in days.

```
to_days()
```

Parameter	Description
	The time interval. The value of this parameter is of the varchar type.

The double type.

Convert 192,848 seconds to a time interval in days.

- Query statement

```
*| SELECT to_days('192848s')
```

- Query and analysis results

to_hours function

The to_hours function converts a time interval to a time interval in hours.

```
to_hours()
```

Parameter	Description
	The time interval. The value of this parameter is of the varchar type.

The double type.

Convert 1.2 days to a time interval in hours.

- Query statement

```
* | SELECT to_hours('1.2d')
```

- Query and analysis results

to_microseconds function

The to_microseconds function converts a time interval to a time interval in microseconds.

```
to_microseconds()
```

Parameter	Description
	The time interval. The value of this parameter is of the varchar type.

The double type.

Convert 3,600 nanoseconds to a time interval in microseconds.

- Query statement

```
* | SELECT to_microseconds('3600ns')
```

- Query and analysis results

to_milliseconds function

The to_milliseconds function converts a time interval to a time interval in milliseconds.

```
to_milliseconds()
```

Parameter	Description
	The time interval. The value of this parameter is of the varchar type.

The double type.

Convert 1.2 seconds to a time interval in milliseconds.

- Query statement

```
* | SELECT to_milliseconds('1.2s')
```

- Query and analysis results

to_minutes function

The to_minutes function converts a time interval to a time interval in minutes.

```
to_minutes()
```

Parameter	Description
	The time interval. The value of this parameter is of the varchar type.

The double type.

Convert 1.2 hours to a time interval in minutes.

- Query statement

```
* | SELECT to_minutes('1.2h')
```

- Query and analysis results

to_most_succinct_time_unit function

The to_most_succinct_time_unit function converts a time interval from the current unit to the optimal unit. The system automatically determines the optimal unit and returns a time interval in the optimal unit.

```
to_most_succinct_time_unit()
```

Parameter	Description
	The time interval. The value of this parameter is of the varchar type.

The varchar type.

Convert 1,340 milliseconds to a time interval in seconds.

- Query statement

```
* | SELECT to_most_succinct_time_unit('1340ms')
```

- Query and analysis results

to_nanoseconds function

The to_nanoseconds function converts a time interval to a time interval in nanoseconds.

```
to_nanoseconds()
```

Parameter	Description
	The time interval. The value of this parameter is of the varchar type.

The double type.

Convert 125 milliseconds to a time interval in nanoseconds.

- Query statement

```
* | SELECT to_nanoseconds('125ms')
```

- Query and analysis results

to_seconds function

The to_seconds function converts a time interval to a time interval in seconds.

```
to_seconds()
```

Parameter	Description
	The time interval. The value of this parameter is of the varchar type.

The double type.

Convert 1,340 milliseconds to a time interval in seconds.

- Query statement

```
* | SELECT to_seconds('1340ms')
```

- Query and analysis results

8.1.27. Window funnel function

Log Service provides the window funnel function. You can use this function to analyze data, such as user behavior, application traffic, and the conversion rate in product promotions. This topic describes the syntax of the window funnel function. This topic also provides examples on how to use the window funnel function.

The following table describes the window funnel function that is supported by Log Service.

Notice

Function	Syntax	Description
window_funnel function	window_funnel(*sliding_window*, *timestamp*, *event_id*, ARRAY[*event_list01*, *event_list02*...])	Searches for an event chain in a sliding time window and counts the maximum number of consecutive events that occurred in the event chain. If the value of the event_id parameter is specified in an event chain, you can use this syntax.
	window_funnel(*sliding_window*, *timestamp*, ARRAY[*event_id*=*event_list01*, *event_id*=*event_list02*...])	Searches for an event chain in a sliding time window and counts the maximum number of consecutive events that occurred in the event chain. If the value of the event_id parameter is not specified in an event chain and you want to use a custom value for the event_id parameter, you can use this syntax for higher flexibility.

Principle

The window funnel function is used to search for an event chain in a sliding time window and count the maximum number of consecutive events in the event chain. The window funnel function starts the count from the first event in the event chain that you specify, checks the events in sequence, and then returns the maximum number of consecutive events.

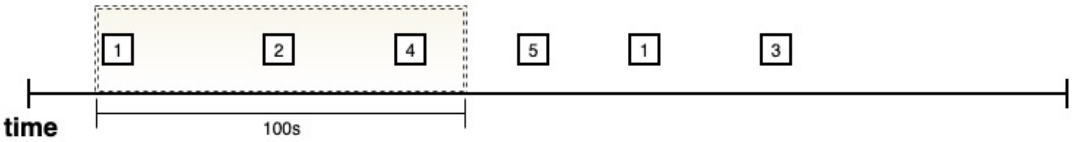
The window funnel function uses the following algorithm:

- The function starts the count from the first event in the event chain and sets the initial value of the event counter to 1. Then, the sliding time window starts.
- In the sliding time window, if the events in the event chain occur in sequence, the event counter is incremented.
- In the sliding time window, if the sequence of events in the event chain is disrupted, the event counter stops. The search stops and a new search starts. The maximum number of consecutive events in the previous search is counted.
- If multiple values of consecutive events exist in the previous search, the function returns the maximum value. The maximum value indicates the maximum number of consecutive events.

For example, you specify a sliding time window of 100 seconds and the following pattern for the event chain: Event 1, Event 2, Event 3, Event 4, and Event 5. However, the events in the event chain occur in the following sequence: Event 1, Event 2, Event 4, Event 5, Event 1, and Event 3. In the sliding time window, the maximum number of consecutive events is 2. A value of 2 indicates the sequential relationship between Event 1 and Event 2.

Notice

- The function must start the count from the first event in the event chain. For example, if the function starts the count in the sequence of Event 2, Event 3, and Event 4, the function returns 0.
- The function must count all events that occur in the event chain. For example, if Event 4 occurs in the sliding time window and Event 3 does not occur in the sliding time window, this search is excluded from the maximum number of consecutive events.



Syntax

The window funnel function supports the following two syntax:

- If the value of the event_id parameter is specified in an event chain, you can use the following syntax:

```
window_funnel(sliding_window, timestamp, event_id, ARRAY[event_list01, event_list02...])
```

- If the value of the event_id parameter is not specified in an event chain and you want to use a custom value for the event_id parameter, you can use the following syntax for higher flexibility:

```
window_funnel(sliding_window, timestamp, ARRAY[event_id=event_list01, event_id=event_list02...])
```

Parameters

Parameter	Description
sliding_window	The sliding time window. Unit: seconds. The value of this parameter is of the bigint type.
timestamp	The timestamp. Unit: seconds. The value of this parameter is of the bigint type. We recommend that you use the built-in __time__ field of Log Service.
event_id	The name of the log field. The value of this parameter is the name of an event. Example: Event A, Event B, or Event C. The value of this parameter is of the varchar type.
event_list	The custom event chain. The event chain can contain up to 32 events. The value of this parameter is of the array type. Examples: • ARRAY['A', 'B', 'C'] • ARRAY[event_id='A', event_id='B', event_id='C']

Examples

An e-commerce store held a promotional activity and analyzed the conversion effect of the activity by using the window funnel function. The conversion process consists of the following three steps: browse the information of a commodity, add the commodity to the online shopping cart, and then purchase the commodity. The following figure shows a sample log that is collected by Log Service.

Log field	Description
behavior_type	The type of user behavior. Valid values: • pv: Browse the information of a commodity. • cart: Add a commodity to the online shopping cart. • buy: Purchase a commodity.
category_id	The category ID of the commodity.
item_id	The ID of the commodity.
timestamp	The point in time at which the user behavior occurred.
user_id	The ID of the user.

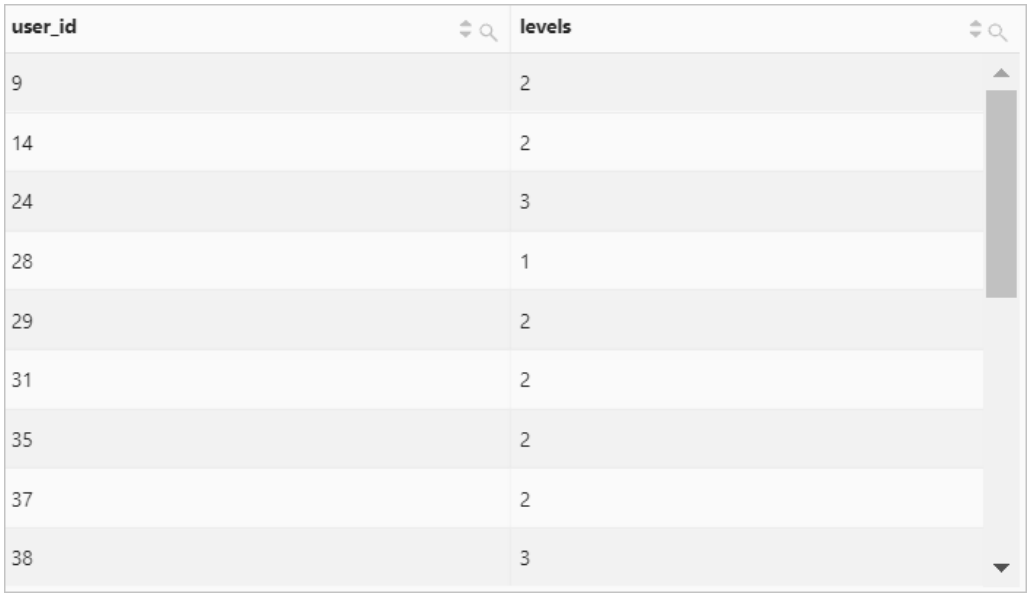
Example 1

Analyze the purchase behavior of users within 24 hours.

- Query statement

```
* |
SELECT
  user_id,
  window_funnel(
    86400,
    timestamp,
    ARRAY [behavior_type='pv', behavior_type='cart',behavior_type='buy']
  ) AS levels
GROUP BY
  user_id
ORDER BY
  user_id
LIMIT
  1000
```

- Query and analysis result
  - The value of the levels field for User 24 is 3. The user completes the purchase in sequence. The user browses the information of a commodity, adds the commodity to the online shopping cart, and then purchases the commodity.
  - The value of the levels field for User 14 is 2. The user browses the information of a commodity and adds the commodity to the online shopping cart. However, the user does not purchase the commodity.



user_id	levels
9	2
14	2
24	3
28	1
29	2
31	2
35	2
37	2
38	3

Example 2

Calculate the number of users for each type of user behavior.

- Query statement

```
* |
SELECT
  levels,
  count,
  sum(count) over(
    ORDER BY
      levels DESC
  ) AS total
FROM  (
    SELECT
      levels,
      count(1) AS count
    FROM      (
        SELECT
          user_id,
          window_funnel(
            86400,
            timestamp,
            ARRAY [behavior_type='pv', behavior_type='cart',behavior_type='buy']
          ) AS levels
        FROM          log
        GROUP BY
          user_id
      )
    GROUP BY
      levels
    ORDER BY
      levels
  )
```

- Query and analysis result
  - The number of users who browse the information of a commodity is 513,194. The number of users who do not continue their purchase process after they browse the information of the commodity is 138,491.

- The number of users who add the commodity to the online shopping cart is 374,703. The number of users who do not continue the purchase process after they add the commodity to the online shopping cart is 198,642.
- The number of users who purchase the commodity is 176,061.

levels	count	total
3	176061	176061
2	198642	374703
1	138491	513194
0	2072	515266

Example 3

Calculate the conversion rate of the preceding promotional activity.

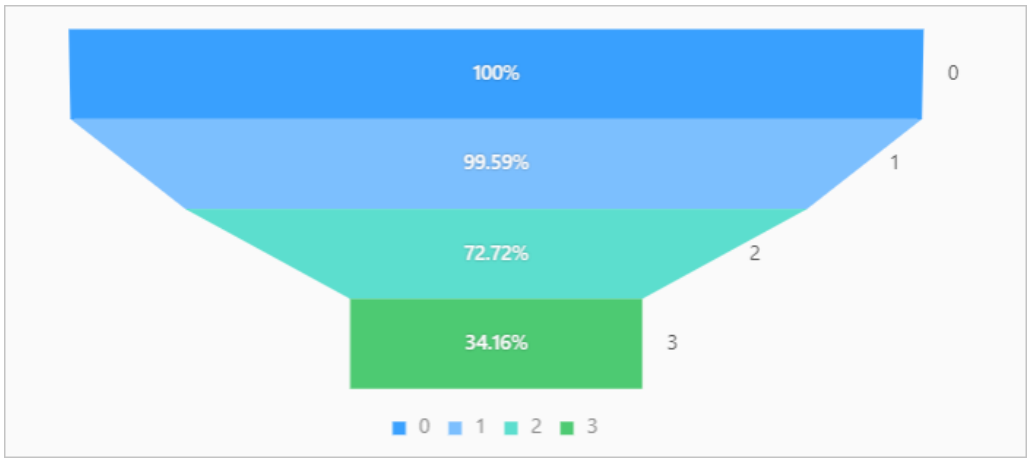
- Absolute conversion rate: the ratio of the number of users who perform a type of user behavior to the total number of users.
- Relative conversion rate: the ratio of the number of users who perform a type of user behavior to the number of users who perform the previous type of user behavior.
- Query statement

```
* |
SELECT
  *,
  100.0 * total /(sum(count) over()) AS "Absolute conversion rate",
  if(
    lag(total, 1, 0) over() = 0,
    100,
    (100.0 * total / lag(total, 1, 0) over())
  ) AS "Relative conversion rate"
FROM  (
    SELECT
      levels,
      count,
      sum(count) over(
        ORDER BY
          levels DESC
      ) AS total
    FROM      (
        SELECT
          levels,
          count(1) AS count
        FROM          (
            SELECT
              user_id,
              window_funnel(
                86400,
                timestamp,
                ARRAY [behavior_type='pv', behavior_type='cart',behavior_type='buy']
              ) AS levels
            FROM              log
            GROUP BY
              user_id
          )
        GROUP BY
          levels
      )
    ORDER BY
      levels
  )
```

- Query and analysis result
  - Table

levels	count	total	Absolute conversion ...	Relative conversion ...
0	2072	515266	100.0	100.0
1	138491	513194	99.59787760108371	99.59787760108371
2	198642	374703	72.72030368780396	73.01390897009708
3	176061	176061	34.168953511390238	46.98681355633662

- Funnel chart



8.1.28. Lambda expressions

Log Service allows you to define a lambda expression in an SQL analytic statement and pass the expression to a specified function. This topic describes the syntax of lambda expressions. This topic also provides examples on how to use the expressions.

Syntax

You must use lambda expressions together with functions, such as filter function, reduce function, transform function, zip_with function, and map_filter function.

```
parameter -> expression
```

Parameter	Description
parameter	The identifier that is used to pass parameters.

expression	The lambda expression, which can include most MySQL expressions. Examples:
	`x -> x + 1`
	`(x, y) -> x + y`
	`x -> regexp_like(x, 'a+')`
	`x -> x[1] / x[2]`
	`x -> if(x > 0, x, -x)`
	`x -> coalesce(x, 0)`
	`x -> cast(x AS JSON)`
	`x -> x + try(1 / 0)`

Examples

This lambda expression is used to return not-null elements in the [5, null, 7, null] array.

- Query statement

```
* | SELECT filter(array[5, null, 7, null], x -> x is not null)
```

- Query and analysis results

This lambda expression is used to return the sum of each element in the [5, 20, 50] array.

- Query statement

```
* | SELECT reduce(array[5, 20, 50], 0, (s, x) -> s + x, s -> s)
```

- Query and analysis results

This lambda expression is used to create a map from two arrays. The values of keys in the map are greater than 10.

- Query statement

```
* | SELECT map_filter(map(array['class01', 'class02', 'class03'], array[11, 10, 9]), (k,v) -> v > 10)
```

- Query and analysis results

This lambda expression is used to transpose elements in two arrays and retrieve elements that are located by using the same index to form a new two-dimensional array.

- Query statement

```
* | SELECT zip_with(array[1, 3, 5], array['a', 'b', 'c'], (x, y) -> (y, x))
```

- Query and analysis results

This lambda expression is used to add 1 to each element in the [5, NULL, 6] array and return the result. The null element in the array is converted to 0 before it is added to 1.

- Query statement

```
* | SELECT transform(array[5, NULL, 6], x -> coalesce(x, 0) + 1)
```

- Query and analysis results

```
* | SELECT filter(array[], x -> true)
* | SELECT map_filter(map(array[],array[]), (k, v) -> true)
* | SELECT reduce(array[5, 6, 10, 20], -- calculates arithmetic average: 10.25
             cast(row(0.0, 0) AS row(sum double, count integer)),
             (s, x) -> cast(row(x + s.sum, s.count + 1) AS row(sum double, count integer)),
             s -> if(s.count = 0, null, s.sum / s.count))
* | SELECT reduce(array[2147483647, 1], cast(0 AS bigint), (s, x) -> s + x, s -> s)
* | SELECT reduce(array[5, 20, null, 50], 0, (s, x) -> s + x, s -> s)
* | SELECT transform(array[array[1, null, 2], array[3, null]], a -> filter(a, x -> x is not null))
* | SELECT zip_with(array['a', 'b', 'c'], array['d', 'e', 'f'], (x, y) -> concat(x, y))
```

Example1: x -> x is not null

Example 2: 0, (s, x) -> s + x, s -> s

Example 3: (k,v) -> v > 10

Example 4: (x, y) -> (y, x)

Example 5: x -> coalesce(x, 0) + 1

Additional examples

8.1.29. Conditional expressions

This topic describes the syntax of conditional expressions and provides examples on how to use conditional expressions.

Expression	Syntax	Description
CASE WHEN statement	CASE WHEN *condition1* THEN *result1* [WHEN *condition2* THEN *result2*] [ELSE *result3*] END	Classifies data based on specified conditions.
IF function	IF(*condition*, *result1*)	If *condition* is evaluated to true, *result1* is returned. Otherwise, null is returned.
	IF(*condition*, *result1*, *result2*)	If *condition* is evaluated to true, *result1* is returned. Otherwise, *result2* is returned.
COALESCE function	COALESCE(*expression1*, *expression2*, *expression3*...)	Returns the first non-null value in multiple expressions.
NULLIF function	NULLIF(*expression1*, *expression2*)	Evaluates whether the values of two expressions are the same. If the values are the same, null is returned. Otherwise, the value of the first expression is returned.
TRY function	TRY(expression)	Captures errors to ensure that Log Service can continue to query and analyze data.

CASE WHEN statement

CASE WHEN statements are used to classify data.

```
CASE WHEN condition1 THEN result1
     [WHEN condition2 THEN result2]
     [ELSE result3]
END
```

Parameter	Description
condition	The conditional expression.
result1	The result that you want to return.

- Example 1: Extract browser information from the value of the http_user_agent field. Then, classify the information into Chrome, Safari, and unknown types and calculate the number of page views (PVs) for the three types.
  - Query statement

```
* |
SELECT
  CASE
    WHEN http_user_agent like '%Chrome%' then 'Chrome'
    WHEN http_user_agent like '%Safari%' then 'Safari'
    ELSE 'unknown'
  END AS http_user_agent,
  count(*) AS pv
GROUP BY
  http_user_agent
```

  - Query and analysis result

http_user_agent	pv
Chrome	5563
Safari	1842
unknown	1666

- Example 2: Query the distribution of requests that are sent at different points in time.
  - Query statement

```
* |
SELECT
  CASE
    WHEN request_time < 10 then 't10'
    WHEN request_time < 100 then 't100'
    WHEN request_time < 1000 then 't1000'
    WHEN request_time < 10000 then 't10000'
    ELSE 'large'
  END AS request_time,
  count(*) AS pv
GROUP BY
  request_time
```

- Query and analysis result

request_time	pv
t100	1563542
large	533

IF function

The IF function is used to classify data. This function works in a similar manner to CASE WHEN statements.

- If *condition* is evaluated to true, *result1* is returned. Otherwise, null is returned.

```
IF(condition, result1)
```

- If *condition* is evaluated to true, *result1* is returned. Otherwise, *result2* is returned.

```
IF(condition, result1, result2)
```

Parameter	Description
condition	The conditional expression.
result	The result that you want to return.

Calculate the ratio of requests whose status code is 200 to all requests.

- Query statement

```
* |
SELECT
  sum(IF(status = 200, 1, 0)) * 1.0 / count(*) AS status_200_percentag
```

- Query and analysis result

status_200_percentage
0.8846858366766299

COALESCE function

The COALESCE function is used to return the first non-null value in multiple expressions.

```
COALESCE(expression1, expression2, expression3...)
```

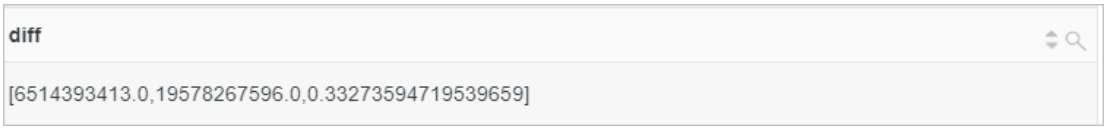
Parameter	Description
expression	The value of this parameter can be an expression of an arbitrary data type.

Calculate the ratio of the expenses of the previous day to the expenses of the same day in the previous month.

- Query statement

```
* |
SELECT
  compare("expenses of the previous day", 604800) AS diff
FROM  (
    SELECT
      COALESCE(sum(PretaxAmount), 0) AS "expenses of the previous day"
    FROM log
  )
```

- Query and analysis result



  - The value 6514393413.0 indicates the expenses of the previous day.
  - The value 19578267596.0 indicates the expenses of the same day in the previous month.
  - The value 0.33273594719539659 indicates the ratio of the expenses of the previous day to the expenses of the same day in the previous month.

NULLIF function

The NULLIF function is used to check whether the values of two columns are the same. If the values are the same, null is returned. Otherwise, the value of the first expression is returned.

```
NULLIF(expression1, expression2)
```

Parameter	Description
expression	The valid scalar expression.

Check whether the values of the client_ip and host fields are the same. If the values are not the same, the value of the client_ip field is returned.

- Query statement

```
* | SELECT NULLIF(client_ip,host)
```

- Query and analysis result

TRY function

The TRY function is used to capture errors to ensure that Log Service can continue to query and analyze data.

```
TRY(expression)
```

Parameter	Description
expression	The value of this parameter can be an expression of an arbitrary data type.

If an error occurs when the regexp_extract function is invoked, the TRY function captures the error. This way, Log Service can continue to query and analyze data. The query and analysis result is returned.

- Query statement

```
* |
SELECT
  TRY(regexp_extract(request_uri, '.*\/(file.*)', 1)) AS file,
  count(*) AS count
GROUP BY
  file
```

- Query and analysis result

file	count
file-5	851
file-7	928
file-3	837
file-4	863

8.2. SQL syntax

8.2.1. EXCEPT clause

The EXCEPT clause is used to combine the result sets of two SELECT statements and return the difference set of the two result sets. The difference set includes the values that are included in the result set of the first SELECT statement but are not included in the result set of the second SELECT statement. This topic describes the syntax of the EXCEPT clause. This topic also provides examples on how to use the EXCEPT clause.

Syntax

```
SELECT key1... FROM logstore1
EXCEPT
SELECT key2... FROM logstore2
```

Notice

- The number and order of the columns in the result sets of the two SELECT statements must be the same. The data types for the columns in the result sets of the two SELECT statements must be the same.
- The EXCEPT clause removes all duplicates from the final results. This way, distinct values are returned in the final results.

Parameters

Parameter	Description
key	The field name, column name, or expression. You can specify different values for *key1* and *key2*, but you must specify the same data types for them.
logstore	The name of the Logstore.

Examples

A Logstore named internal-diagnostic_log is used to store important logs. The important logs record information about the log consumption latency, alerts, and log collection of each Logstore. A Logstore named internal-operation_log is used to store detailed logs. The detailed logs record information about all the operations on resources in a project. You can use the EXCEPT clause to query which Logstores have detailed logs but not important logs.

- Query statement

```
* |
SELECT
  logstore
FROM  internal-operation_log
EXCEPT
SELECT
  logstore
FROM  internal-diagnostic_log
```

- Query and analysis results

8.2.2. EXISTS clause

An EXISTS clause is used to check whether a subquery returns a specific result. If the subquery in an EXISTS clause returns a specific result, true is returned and the outer SQL statement is executed.

Syntax

```
* | SELECT...FROM...WHERE EXISTS (subquery)
```

Parameters

Parameter	Description
subquery	The value of this parameter is a SELECT statement.

Example

Check whether the read and write latency of a specific Logstore is greater than 1,000 microseconds. If the latency is greater than 1,000 microseconds, the information of the related consumer group is returned.

- Query statement

```
* |
SELECT
  consumer_group
FROM  "internal-diagnostic_log"
WHERE
  EXISTS (
    SELECT
     Latency
    FROM     internal-operation_log
    WHERE
      "internal-diagnostic_log".LogStore = "internal-operation_log".logstore and latency >1000
  )
```

- Query and analysis result

8.2.3. GROUP BY clause

The GROUP BY clause is used together with aggregate functions to group analysis results based on one or more columns that you specify. The GROUP BY clause can also be used together with ROLLUP, CUBE, and GROUPING SETS to generate multiple grouping sets.

Syntax

- GROUP BY

  The GROUP BY clause groups analysis results based on one or more columns that you specify.

```
SELECT
 key,
  ...
 aggregate function
 GROUP BY
 key,...
```

- GROUP BY ROLLUP

  The GROUP BY ROLLUP clause groups analysis results based on the rollup operation. The clause returns a subtotal for each group and a grand total for all groups. For example, if you use GROUP BY ROLLUP (a, b), the following grouping sets are produced: (a, b), (a, null), and (null, null).

```
SELECT
key,
 ...
aggregate function
GROUP BY ROLLUP (key,...)
```

- GROUP BY CUBE

  The GROUP BY CUBE clause groups analysis results based on all possible combinations of columns. For example, if you use GROUP BY CUBE (a, b), the following grouping sets are produced: (a, b), (null, b), (a, null), and (null, null).

```
SELECT
key,
...
aggregate function
GROUP BY CUBE (key,...)
```

- GROUP BY GROUPING SETS

  The GROUP BY GROUPING SETS clause groups analysis results based on the columns that you specify in sequence. For example, if you use GROUP BY GROUPING SETS (a, b), the following grouping sets are produced: (a, null) and (null, b).

```
SELECT
key,
...
aggregate function
GROUP BY GROUPING SETS (key,...)
```

Notice If you use the GROUP BY clause in an analytic statement, the system can query only a column that is included in the GROUP BY clause or perform aggregation on an arbitrary column when the system executes the SELECT statement. For example, `* | SELECT status, request_time, COUNT(*) AS PV GROUP BY status` is invalid because the request_time column is not included in the GROUP BY clause. You can change the statement to `* | SELECT status, arbitrary(request_time), count(*) AS PV GROUP BY status`, which is valid.

Parameters

Parameter	Description
key	The name of the log field or the name of the column whose values are returned by an aggregate function. The GROUP BY clause groups results based on the log field or column that you specify. The GROUP BY clause allows you to specify one or more columns.
aggregate function	The aggregate function that is used together with the GROUP BY clause. The GROUP BY clause is often used together with aggregate functions, such as min, max, avg, sum, and count. For more information, see Aggregate function.

Examples

Example 1

Group the numbers of page views (PVs) based on status codes.

- Query statement

```
* | SELECT status, count(*) AS PV GROUP BY status
```

- Query and analysis results

status	PV
205	59
301	41
204	48
402	30
202	68
307	66
401	31
203	64

Example 2

Group the numbers of PVs based on 1-hour intervals. In the following statement, the __time__ field is a reserved field in Log Service. This field indicates the time column. time is the alias of `date_trunc('hour', __time__)`. For more information about the date_trunc function, see date_trunc function.

- Query statement

```
* |
SELECT
  count(*) AS PV,
  date_trunc('hour', __time__) AS time
GROUP BY
  time
ORDER BY
  time
LIMIT
  1000
```

- Query and analysis results

PV	time
1202	2021-08-10 00:00:00.000
10159	2021-08-10 01:00:00.000
28001	2021-08-10 02:00:00.000

Example 3

Group the numbers of PVs based on 5-minute intervals.

- Query statement

  The date_trunc function allows you to measure statistics only based on a fixed interval. The interval is determined by the time unit that you specify in the function. If you want to measure statistics based on a custom interval, we recommend that you perform a mathematical modulo operation to group data. In the following statement, %300 specifies that a mathematical modulo operation is performed to group data at 5-minute intervals.

```
* |
SELECT
  count(*) AS PV,
  __time__-__time__ % 300 AS time
GROUP BY
  time
LIMIT
  1000
```

- Query and analysis results

PV	time
143	1628525100
31	1628526600
44	1628526900

Example 4

Group the numbers of PVs based on the request_method and status columns. The GROUP BY GROUPING SETS clause calculates the numbers of PVs first for each request method and then for each status.

- Query statement

```
* |
SELECT
  request_method,
  status,
  count(*) AS PV
GROUP BY
  GROUPING SETS (request_method, status)
```

- Query and analysis results

request_method	status	PV
GET	null	285
POST	null	51
DELETE	null	28
PUT	null	60
null	200	382
null	204	2
null	501	2
null	202	4
null	305	4

Example 5

Group the numbers of PVs based on the request_method and status columns. The following grouping sets are produced: (null, null), (request_method, null), (null, status), and (request_method, status). The GROUP BY CUBE clause calculates the numbers of PVs for each group.

- Query statement

```
* |
SELECT
  request_method,
  status,
  count(*) AS PV
GROUP BY
 CUBE (request_method, status)
```

- Query and analysis results

request_method	status	PV
null	null	341
GET	null	215
PUT	null	53
DELETE	null	27
POST	null	45
HEAD	null	1
null	200	296
null	204	2
null	304	1

Total:58 < 1 / 3 >

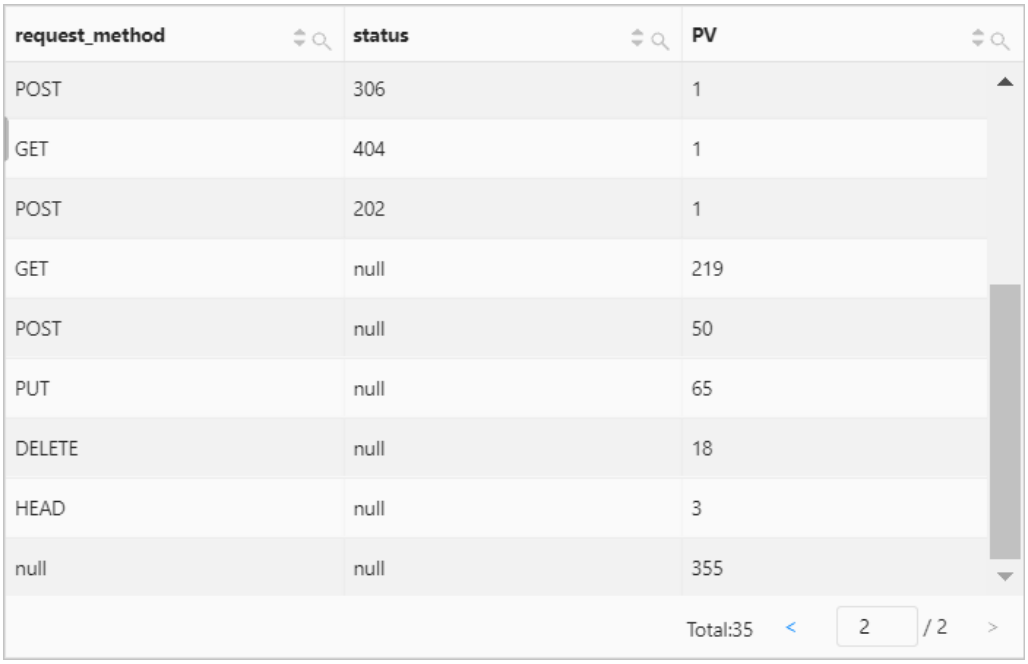
Example 6

Group the numbers of PVs based on the request_method and status columns. The following grouping sets are produced: (request_method, status), (request_method, null), and (null, null). The GROUP BY ROLLUP clause calculates the numbers of PVs for each group.

- Query statement

```
* |
SELECT
  request_method,
  status,
  count(*) AS PV
GROUP BY
 ROLLUP (request_method, status)
```

- Query and analysis results



request_method	status	PV
POST	306	1
GET	404	1
POST	202	1
GET	null	219
POST	null	50
PUT	null	65
DELETE	null	18
HEAD	null	3
null	null	355

Total:35 < 2 / 2 >

8.2.4. HAVING clause

The HAVING clause is used to specify filter conditions for the results that are returned by GROUP BY clauses or aggregate functions.

Syntax

```
HAVING bool_expression
```

Notice

- The HAVING clause is used to filter results that are returned by GROUP BY clauses or aggregate functions. The WHERE clause is used to filter raw data before the data is aggregated.
- The HAVING clause is used before the ORDER BY clause and after the GROUP BY clause.

Parameters

Parameter	Description
bool_expression	The Boolean expression.

Examples

- Example 1: Return the request URIs whose average request duration is longer than 40 seconds.

- Query statement

```
* |
SELECT
  avg(request_time) AS avg_time,
  request_uri
GROUP BY
  request_uri
HAVING
  avg(request_time) > 40
```

- Query and analysis results

avg_time	request_uri
45.03659584919563	/request/path-1/file-0
45.388778550148959	/request/path-3/file-6
45.269188395152408	/request/path-2/file-4
44.91338974614236	/request/path-0/file-0
44.924772223590249	/request/path-2/file-3
45.09470581009704	/request/path-0/file-3
45.013995215311009	/request/path-3/file-4
45.099331306990887	/request/path-3/file-8
44.946835443037979	/request/path-2/file-2

总数：40 < 1 / 2 >

- Example 2: Query the write latency of projects in service logs and return the projects whose write latency is greater than 1,000 microseconds.
  - Query statement

```
method: PostLogstoreLogs |
SELECT
  avg(latency) AS avg_latency,
  Project
GROUP BY
  Project
HAVING
  avg_latency > 1000
```

  - Query and analysis results

avg_latency	Project
1569.909090909091	datala[redacted]engdu

8.2.5. INSERT INTO clause

You can use an INSERT INTO clause to write query and analysis results from one Logstore to another Logstore. The two Logstores belong to the same project.

Syntax

```
INSERT INTO target_logstore (key)
SELECT key FROM source_logstore
```

Notice

- In the destination Logstore, you must create an index for the *key* field that you specify and enable the analysis feature for the field.
- *target_logstore* must be followed by a field that you want to write to the destination Logstore. For example, `* | INSERT INTO target_logstore SELECT...` is an invalid statement.
- If the data types of the specified fields do not match the data types that are supported by Log Service, you must specify a data type conversion function in the SELECT statement to convert the data types of the fields. For more information, see Data type conversion functions.
- You can use an INSERT INTO clause to write up to 10,000 data entries to a destination Logstore at a time.
- The source and destination Logstores must reside in mainland China.

Parameters

Parameter	Description
target_logstore	The name of the destination Logstore. **Note** The destination Logstore must be different from the source Logstore.
source_logstore	The name of the source Logstore.
key	The field name or column name.

Example

Calculate the number of page views (PVs) for each status code in a Logstore named website_log and write the query and analysis result to a Logstore named test_insert.

Notice Before you execute the following query statement, you must create indexes for the status and PV fields in the test_insert Logstore and enable the analysis feature for the fields.

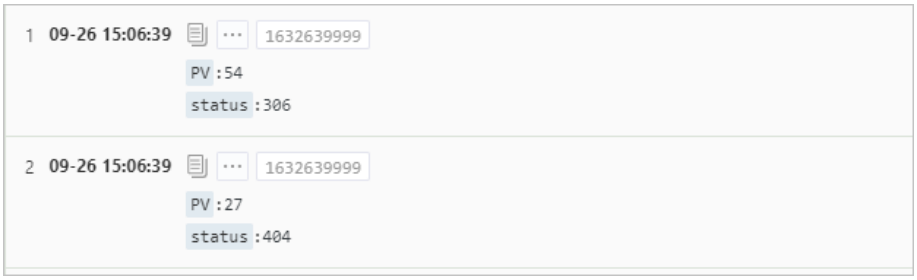
- Query statement

```
* | INSERT INTO test_insert(status,PV) SELECT status, count(*) AS PV FROM website_log GROUP BY sta
tus
```

- Query and analysis result (source Logstore)

rows	info
22	

- Query and analysis result (destination Logstore)



8.2.6. INTERSECT clause

The INTERSECT clause is used to combine the result sets of two SELECT statements and return only rows that are common to the result sets of the two SELECT statements. This topic describes the syntax of the INTERSECT clause. This topic also provides examples on how to use the INTERSECT clause.

Syntax

```
SELECT key1... FROM logstore1
INTERSECT
SELECT key2... FROM logstore2
```

Notice

- The number and order of the columns in the result sets of the two SELECT statements must be the same. The data types for the columns in the result sets of the two SELECT statements must be the same.
- The INTERSECT clause removes all duplicates from the final results. This way, distinct values are returned in the final results.

Parameters

Parameter	Description
key	The field name, column name, or expression. You can specify different values for *key1* and *key2*, but you must specify the same data types for them.
logstore	The name of the Logstore.

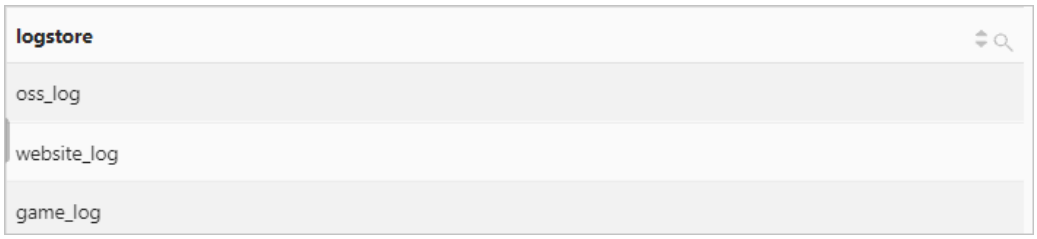
Examples

A Logstore named internal-diagnostic_log is used to store important logs. The important logs record information about the log consumption latency, alerts, and log collection of each Logstore. A Logstore named internal-operation_log is used to store detailed logs. The detailed logs record information about all the operations on resources in a project. You can use the INTERSECT clause to query which Logstores have both detailed logs and important logs.

- Query statement

```
* |
SELECT
  logstore
FROM  internal-operation_log
INTERSECT
SELECT
  logstore
FROM  internal-diagnostic_log
```

- Query and analysis results



8.2.7. JOIN clause

You can specify JOIN clauses in SQL statements to join multiple tables based on the fields that are shared by the tables. Log Service allows you to join data that is stored in different Logstores. You can also join data that is stored in a Logstore with data that is stored in a MySQL database or with data that is stored in an Object Storage Service (OSS) bucket. This topic describes the syntax of JOIN clauses and provides examples on how to use JOIN clauses.

Syntax

```
SELECT table.key
FROM table1
INNER|LEFT|RIGHT|FULL OUTER JOIN table2
ON table1.key=table2.key
```

Log Service allows you to use INNER JOIN clauses, LEFT JOIN clauses, RIGHT JOIN clauses, and OUTER JOIN clauses in SELECT statements. For more information, see JOIN.

JOIN syntax	Description
INNER JOIN	Returns only the matching rows that meet the conditions specified in the SELECT statement between two tables.
LEFT JOIN	Returns all rows that meet the conditions specified in the SELECT statement from the left table (table1) even if no matching rows exist in the right table (table2).
RIGHT JOIN	Returns all rows that meet the conditions specified in the SELECT statement from the right table (table2) even if no matching rows exist in the left table (table1).
FULL OUTER JOIN	Returns the rows that meet the conditions specified in the SELECT statement if a table contains a matching row.

Parameters

Parameter	Description
key	A log field or an expression. The value of this parameter can be of an arbitrary data type.
table	*table1* is a Logstore and *table2* can be a Logstore, a MySQL database, or an OSS bucket. For more information, see Associate Log Service with a MySQL database and Associate Log Service with an OSS bucket.

Examples

A Logstore named internal-diagnostic_log is used to record the logs that include information, such as the consumption latency, alerts, and log collection for each Logstore in a project. A Logstore named internal-operation_log is used to record the operation logs of all resources in the project. You can use a JOIN clause to query log data from the two Logstores and obtain the information about the consumer groups, consumption latency, and request methods for each Logstore in the project.

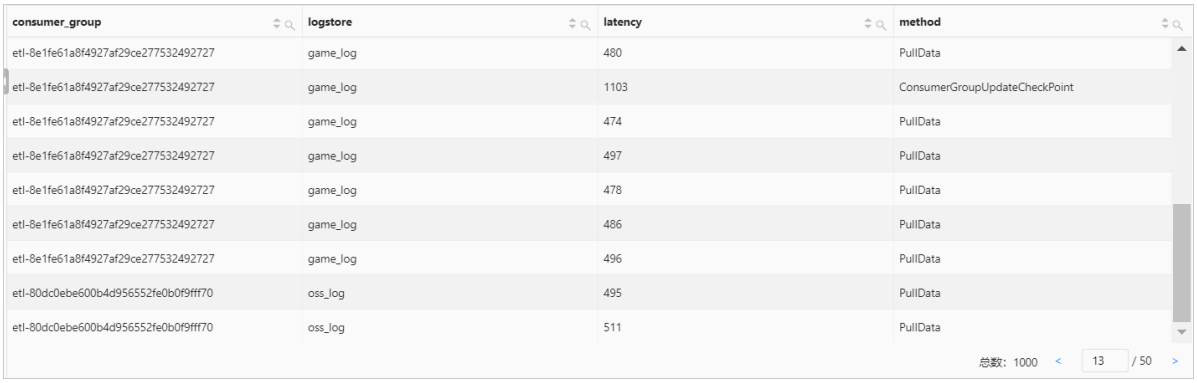
Example 1: INNER JOIN

- Query statement

```
* |
SELECT
  "internal-diagnostic_log".consumer_group,
  "internal-diagnostic_log".logstore,
  "internal-operation_log".Latency,
  "internal-operation_log".Method
FROM  "internal-diagnostic_log"
  INNER JOIN "internal-operation_log" ON "internal-diagnostic_log".logstore = "internal-operation_
log".logstore
LIMIT
  10000
```

- Query and analysis result

  In this example, 1,328 rows of data that meet the specified conditions are returned.



consumer_group	logstore	latency	method
etl-8e1fe61a8f4927af29ce277532492727	game_log	480	PullData
etl-8e1fe61a8f4927af29ce277532492727	game_log	1103	ConsumerGroupUpdateCheckPoint
etl-8e1fe61a8f4927af29ce277532492727	game_log	474	PullData
etl-8e1fe61a8f4927af29ce277532492727	game_log	497	PullData
etl-8e1fe61a8f4927af29ce277532492727	game_log	478	PullData
etl-8e1fe61a8f4927af29ce277532492727	game_log	486	PullData
etl-8e1fe61a8f4927af29ce277532492727	game_log	496	PullData
etl-80dc0ebe600b4d956552fe0b0f9fff70	oss_log	495	PullData
etl-80dc0ebe600b4d956552fe0b0f9fff70	oss_log	511	PullData

总数: 1000 < 13 / 50 >

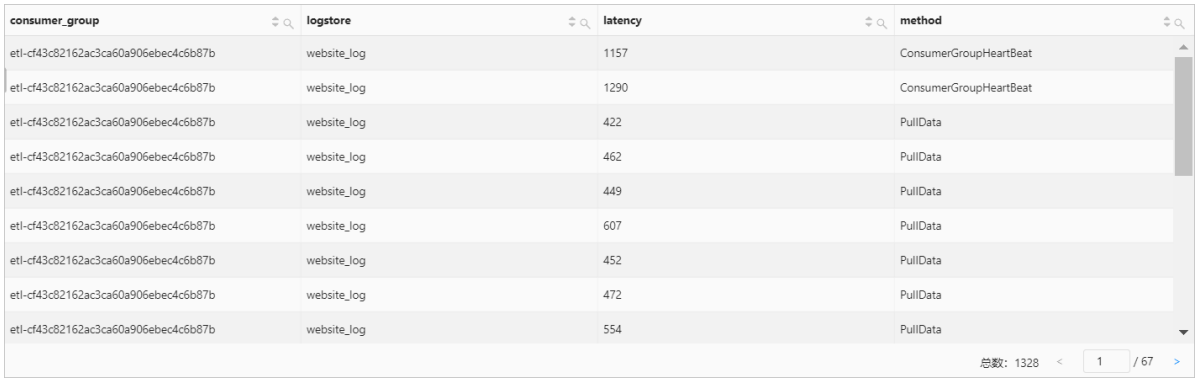
Example 2: LEFT JOIN

- Query statement

```
* |
SELECT
  "internal-diagnostic_log".consumer_group,
  "internal-diagnostic_log".logstore,
  "internal-operation_log".Latency,
  "internal-operation_log".Method
FROM  "internal-diagnostic_log"
  LEFT JOIN "internal-operation_log" ON "internal-diagnostic_log".logstore = "internal-operation_l
og".logstore
LIMIT
  10000
```

- Query and analysis result

  In this example, 1,328 rows of data in the internal-diagnostic_log Logstore are returned.



consumer_group	logstore	latency	method
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	1157	ConsumerGroupHeartBeat
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	1290	ConsumerGroupHeartBeat
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	422	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	462	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	449	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	607	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	452	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	472	PullData
etl-cf43c82162ac3ca60a906ebec4c6b87b	website_log	554	PullData

总数：1328 < 1 / 67 >

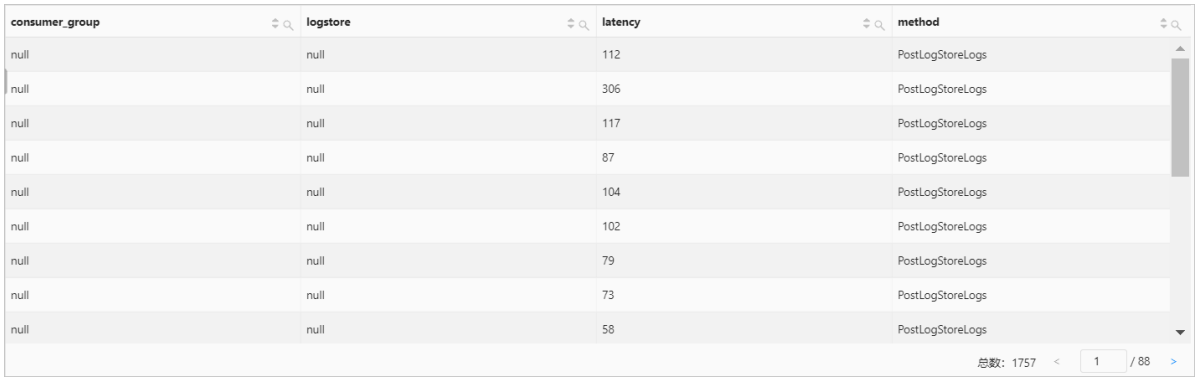
Example 3: RIGHT JOIN

- Query statement

```
* |
SELECT
  "internal-diagnostic_log".consumer_group,
  "internal-diagnostic_log".logstore,
  "internal-operation_log".Latency,
  "internal-operation_log".Method
FROM  "internal-diagnostic_log"
  RIGHT JOIN "internal-operation_log" ON "internal-diagnostic_log".logstore = "internal-operation_
log".logstore
LIMIT
  10000
```

- Query and analysis result

  In this example, 1,757 rows of data in the internal-operation_log Logstore are returned.



consumer_group	logstore	latency	method
null	null	112	PostLogStoreLogs
null	null	306	PostLogStoreLogs
null	null	117	PostLogStoreLogs
null	null	87	PostLogStoreLogs
null	null	104	PostLogStoreLogs
null	null	102	PostLogStoreLogs
null	null	79	PostLogStoreLogs
null	null	73	PostLogStoreLogs
null	null	58	PostLogStoreLogs

总数：1757 < 1 / 88 >

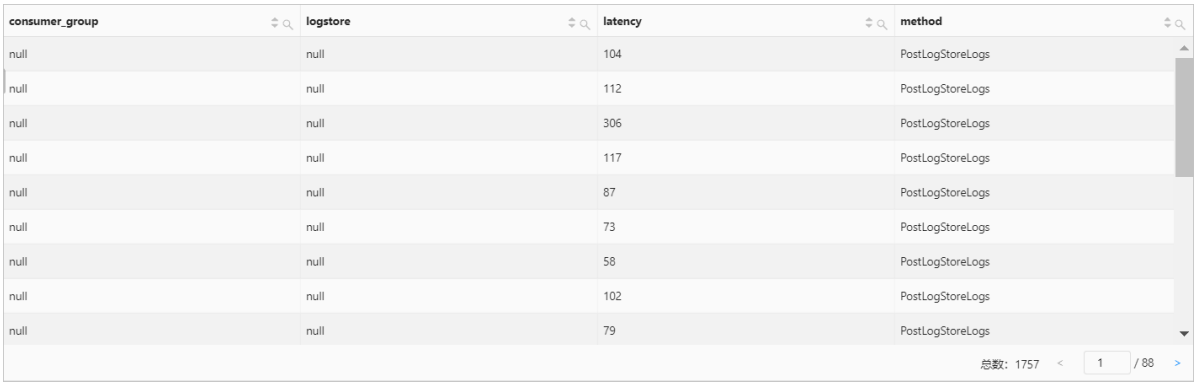
Example 4: FULL OUTER JOIN

- Query statement

```
* |
SELECT
  "internal-diagnostic_log".consumer_group,
  "internal-diagnostic_log".logstore,
  "internal-operation_log".Latency,
  "internal-operation_log".Method
FROM  "internal-diagnostic_log"
  FULL OUTER JOIN "internal-operation_log" ON "internal-diagnostic_log".logstore = "internal-opera
tion_log".logstore
LIMIT
  10000
```

- Query and analysis result

  In this example, 1,757 rows of data that meet the specified conditions are returned.



consumer_group	logstore	latency	method
null	null	104	PostLogStoreLogs
null	null	112	PostLogStoreLogs
null	null	306	PostLogStoreLogs
null	null	117	PostLogStoreLogs
null	null	87	PostLogStoreLogs
null	null	73	PostLogStoreLogs
null	null	58	PostLogStoreLogs
null	null	102	PostLogStoreLogs
null	null	79	PostLogStoreLogs

总数: 1757 < 1 / 88 >

8.2.8. LIMIT clause

By default, Log Service returns 100 rows of data in the query and analysis results. You can use a LIMIT clause to specify the number of rows that can be returned.

Syntax

Log Service supports the following types of LIMIT clauses:

- The following LIMIT clause returns the first x rows of data in the query and analysis results:

```
LIMIT
```

- The following LIMIT clause returns x rows of data starting from the y row in the query and analysis results:

```
LIMIT ,
```

> **Notice**
>
> - The LIMIT clause is used to obtain final results rather than SQL intermediate results.
> - You cannot use LIMIT clauses in subqueries. For example, the following statement is invalid: `* | select count(1) from ( select distinct(url) from limit 0,1000)` .

Parameters

Parameter	Description
	The number of rows that can be returned. • If you use `LIMIT `, the valid values of are [0,1000000]. • If you use `LIMIT ,`, the valid values of are [0,10000].
	The offset. Valid values: [0,1000000].

Notice The sum of and cannot exceed 1,000,000.

Examples

- Return the first 200 rows of data in the query and analysis results.
  - Query statement

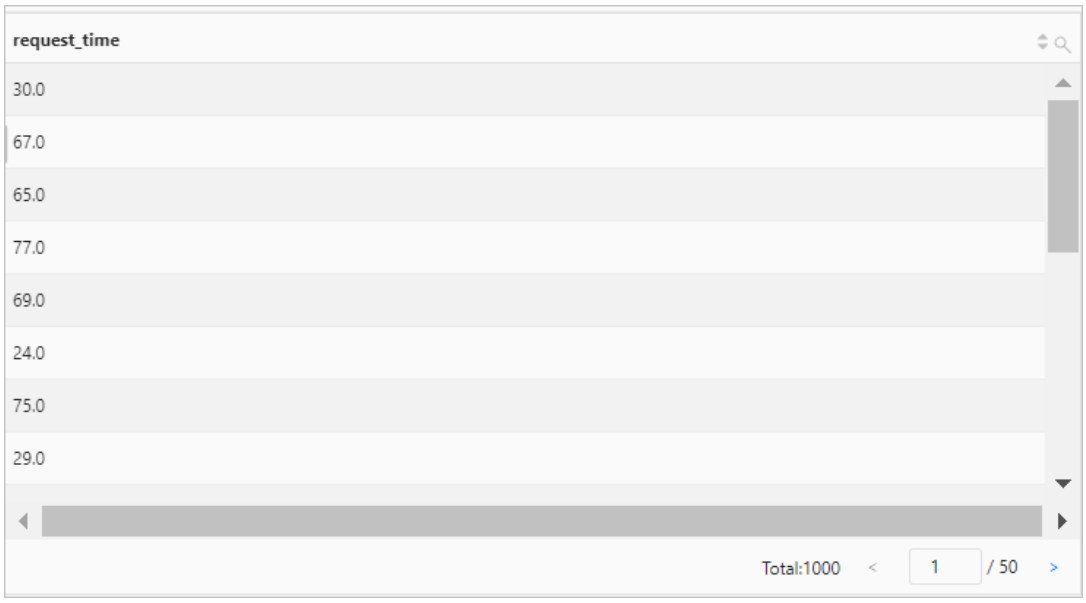
```
* | SELECT  request_time LIMIT 200
```

  - Query and analysis results

- Return 1,000 rows of data with an offset of 100 in the query and analysis results.
  - Query statement

```
* | SELECT  request_time LIMIT 100,1000
```

- Query and analysis results



- Return the top three request URIs with the longest request durations.
  - Query statement

```
* |
SELECT
  request_uri AS top_3,
  request_time
ORDER BY
  request_time DESC
LIMIT
  3
```

  - Query and analysis results

top_3	request_time
/request/path-3/file-2	80.0
/request/path-2/file-4	80.0
/request/path-0/file-8	79.0

8.2.9. ORDER BY clause

The ORDER BY clause is used to sort query and analysis results based on specified column names.

Syntax

```
ORDER BY Column name [DESC | ASC]
```

Note

- You can specify multiple column names to sort data in different orders. Example: `ORDER BY Column Name 1 [DESC | ASC], Column Name 2 [DESC | ASC]`.
- If you do not specify the DESC or ASC keyword, the system sorts the query and analysis results in ascending order by default.
- If a specified column has duplicate values, the sorting results may vary each time the query and analysis results are sorted. If you want to ensure consistent sorting results, you can specify multiple columns for sorting.

Parameters

Parameter	Description
Column name	The name of the log field or the name of the column whose values are returned by an aggregate function. The ORDER BY clause sorts results based on the log field or column that you specify.
DESC	Data is sorted in descending order.
ASC	Data is sorted in ascending order.

Examples

- Example 1: Count the numbers of requests that correspond to different HTTP status codes and sort the query and analysis results in descending order by the numbers.
  - Query statement

```
* |
SELECT
  count(*) AS PV,
  status
GROUP BY
  status
ORDER BY
  PV DESC
```

  - Query and analysis results

PV	status
163135	200
1224	206
1186	207
1185	305
1184	301
1182	307
1180	302
1177	203

- Example 2: Calculate the average write latency of each Logstore and sort the query and analysis results in descending order by the average latencies.
  - Query statement

```
method :PostLogstoreLogs |
SELECT
  avg(latency) AS avg_latency,
  LogStore
GROUP BY
  LogStore
ORDER BY
  avg_latency DESC
```

  - Query and analysis results

avg_latency	Logstore
3833.0	test
2691.133333333333	website_log
2608.0555555555557	date

- Example 3: Count the numbers of requests that correspond to different request durations and sort the query and analysis results in ascending order by the request durations.

  In the following query statement, content, time, and request_time are fields in JSON logs.

> **Notice**
>
> When you query and analyze JSON logs, make sure that the following requirements are met. For more information, see Query and analyze JSON logs.
>
> - You must add the parent path to a field name in JSON logs. Example: content.time.request_time.
> - You must use double quotation marks ("") to enclose a field name in JSON logs in an analytic statement. Example: "content.time.request_time".

  - Query statement

```
* |
SELECT
  "content.time.request_time",
  count(*) AS count
GROUP BY
  "content.time.request_time"
ORDER BY
  "content.time.request_time"
```

  - Query and analysis results

content.time.request_time	count
10.0	145
11.0	123
12.0	113

8.2.10. UNION clause

A UNION clause is used to combine the analysis results of multiple SELECT statements.

Syntax

```
SELECT key1 FROM logstore1
UNION
SELECT key2 FROM logstore2
UNION
SELECT key3 FROM logstore3
```

Notice Each SELECT statement in a UNION clause must have the same number of columns. The values of the columns in the same position must be of the same data type.

Parameters

Parameter	Description
key	The field name or column name. The values of the *key1*, *key2*, and *key3* parameters must be of the same data type. You can specify different field names or column names.
logstore	The name of the Logstore.

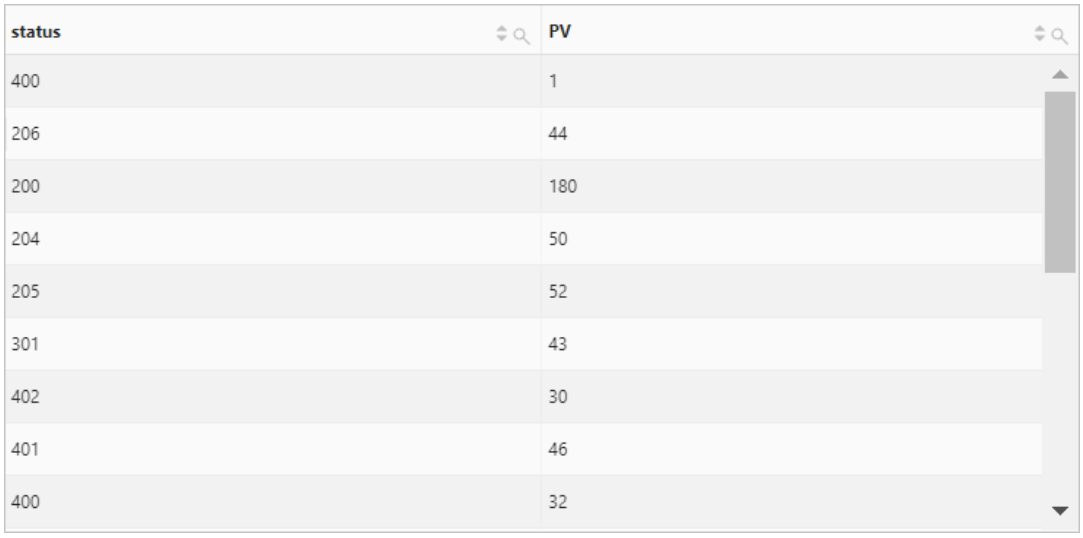
Example

Calculate the number of the page views (PVs) for each status code from the website_log Logstore and the internal-operation_log Logstore. All queried and analyzed data is combined and returned at the same time.

- Query statement

```
* |
SELECT
  status,
  count(*) AS PV
FROM  website_log
GROUP BY
  status
UNION
SELECT
  status,
  count(*) AS PV
FROM  internal-operation_log
GROUP BY
  status
```

- Query and analysis result



status	PV
400	1
206	44
200	180
204	50
205	52
301	43
402	30
401	46
400	32

8.2.11. UNNEST clause

In complex business scenarios, the value of a log field may be of a complex data type, such as array or map. If you want to query and analyze logs that contain fields whose values are of the preceding types, you can use an UNNEST clause to expand the field values into multiple rows for analysis.

Syntax

- Expand an array into multiple rows. *column_name* specifies the column name of the rows.

```
UNNEST() AS table_alias(column_name)
```

- Expands a map into multiple rows. *key_name* specifies the column name of the keys and *value_name* specifies the column name of the values.

```
unnest() AS table(key_name,value_name)
```

Notice You can use an UNNEST clause to expand only arrays or maps. If you want to expand a string, you must convert the string to JSON data. Then, you can use the `try_cast(json_parse(array_column) as array(bigint))` syntax to convert the JSON data to an array or a map. For more information, see Data type conversion functions.

Parameters

Parameter	Description
	The value of this parameter is an array.
column_name	The column name that you specify for the data expanded from the array. This column is used to store the elements in the array.
	The value of this parameter is a map.
key_name	The column name that you specify for the data expanded from the map. This column is used to store the keys in the map.
value_name	The column name that you specify for the data that is expanded from the map. This column is used to store the values in the map.

Examples

Example 1:

Expand the value of the number field into multiple rows. The field value is an array.

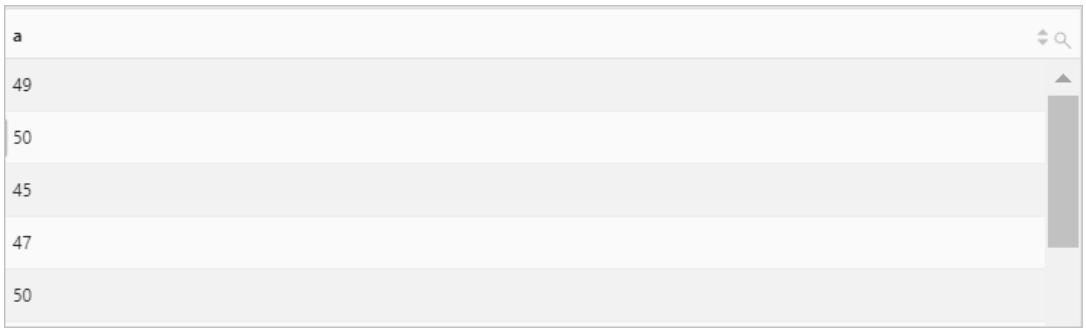
- Sample field

```
number:[49, 50, 45, 47, 50]
```

- Query statement

```
* |
SELECT
  a
FROM  log,
  unnest(cast(json_parse(number) AS array(bigint))) AS t(a)
```

- Query and analysis result



Example 2:

Expand the value of the number field into multiple rows and calculate the sum of the elements. The field value is an array.

- Sample field

```
number:[49, 50, 45, 47, 50]
```

- Query statement

```
* |
SELECT
  sum(a) AS sum
FROM  log,
  unnest(cast(json_parse(number) as array(bigint))) AS t(a)
```

- Query and analysis result

Example 3

Expand the value of the number field into multiple rows and perform the GROUP BY operation on the elements. The field value is an array.

- Sample field

```
number:[49, 50, 45, 47, 50]
```

- Query statement

```
* |
SELECT
  a, count(*) AS count
FROM  log,
  unnest(cast(json_parse(number) as array(bigint))) AS t(a) GROUP BY a
```

- Query and analysis result

a	count
50	1194
47	597
49	597
45	597

Example 4

Expand the value of the number field into multiple rows. The field value is a map.

- Sample field

```
result:{
  anomaly_type:"OverThreshold"
  dim_name:"request_time"
  is_anomaly:true
  score:1
  value:"3.000000"}
```

- Query statement

```
* |
select
  key,
  value
FROM  log,
  unnest(
    try_cast(json_parse(result) as map(varchar, varchar))
  ) as t(key, value)
```

- Query and analysis result

key	value
anomaly_type	OverThreshold
dim_name	request_time
is_anomaly	true
score	1
value	33.000000

Example 5

Expand the value of the number field into multiple rows and perform the GROUP BY operation on each key. The field value is a map.

- Sample field

```
result:{
  anomaly_type:"OverThreshold"
  dim_name:"request_time"
  is_anomaly:true
  score:1
  value:"3.000000"}
```

- Query statement

```
* |
select
  key,
  count(*) AS count
FROM  log,
  unnest(
    try_cast(json_parse(result) as map(varchar, varchar))
  ) as t(key, value)
GROUP BY
  key
```

- Query and analysis result

key	count
anomaly_type	5422
dim_name	5422
is_anomaly	5422
score	5422
value	5422

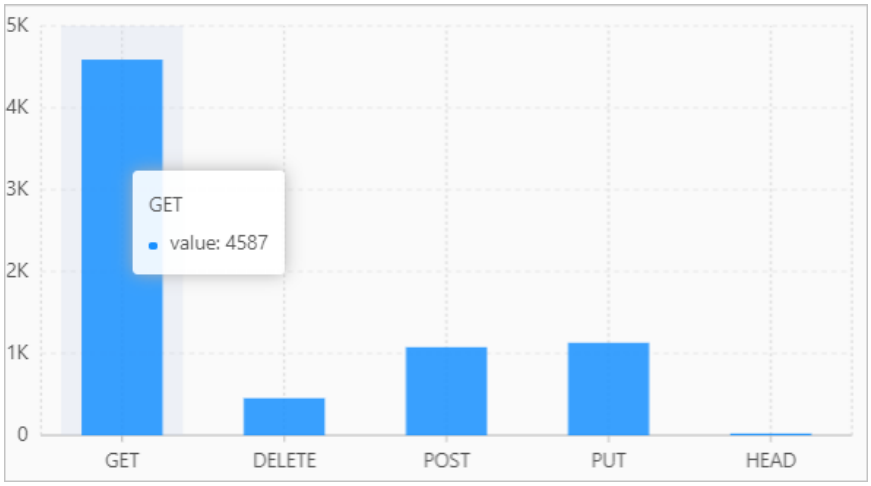
Example 6

Invoke the histogram function to obtain the number of requests that are sent by using each request method. The return value is a map. Then, use an UNNEST clause to expand the map into multiple rows and display the query and analysis result on a column chart.

- Query statement

```
* |
SELECT
  key,
  value
FROM(
    SELECT
      histogram(request_method) AS result
    FROM      log
  ),
  unnest(result) AS t(key, value)
```

- Query and analysis result



8.2.12. VALUES clause

The VALUES clause is used to insert a small amount of temporary data into a table for query and analysis. This topic describes the syntax of the VALUES clause. This topic also provides examples on how to use the VALUES clause.

Syntax

```
VALUES(column_value01, column_value02...) table_name(column_name01,column_name02...)
```

Parameters

Parameter	Description
column_value	The values that you want to insert into the column. You can specify constants, expressions, or functions.
table_name	The name of the table into which you want to insert the values.
column_name	The name of the column into which you want to insert the values.

Examples

Use the VALUES clause to insert data into a column named pv in the table named access.

- Query statement

```
* |
SELECT
  pv
FROM  (
    VALUES
      (0),
      (1),
      (2),
      (3),
      (4),
      (5),
      (6),
      (7),
      (8),
      (9)
  ) AS access(pv)
```

- Query and analysis results

8.2.13. WITH clause

You can use a WITH clause to save the result of a subquery to a temporary table. Then, you can execute an SQL statement to analyze the data in the temporary table. You can use WITH clauses to simplify SQL statements and improve readability. This topic describes the syntax of WITH clauses and provides examples on how to use WITH clauses.

Syntax

```
WITH table_name AS (select_statement) select_statement
```

Parameters

Parameter	Description
table_name	The name of the temporary table.
select_statement	The complete SELECT statement.

Example

Analyze the average request length for each host in a Logstore named website_log and save the analysis result to a table named T1. Analyze the average request length for each host in a Logstore named access_log and save the analysis result to a table named T2. Then, use a JOIN clause to combine T1 and T2 and query the average request length for each host that is contained in both tables.

- Query statement

```
* | with T1 AS (
  SELECT
    host,
    avg(request_length) length
  FROM    website_log
  GROUP BY
    host
),
T2 AS (
  SELECT
    host,
    avg(request_length) length
  FROM    access_log
  GROUP BY
    host
)
SELECT
  T1.host,
  T1.length,
  T2.length
FROM  T1
  JOIN T2 ON T1.host = T2.host
```

- Query and analysis result

host	length	length
www.l[redacted]com	4150.775370581528	4150.775370581528
www.t[redacted]com	3767.3465346534654	3767.3465346534654
www.v[redacted]k.com	4252.855555555556	4252.855555555556
www.l[redacted].com	3665.181818181818	3665.181818181818
www.e[redacted]com	4227.140536149472	4227.140536149472
www.x[redacted]com	4197.650477707007	4197.650477707007
www.l[redacted].com	3556.4367816091954	3556.4367816091954
www.c[redacted].com	4124.819313466616	4124.819313466616

8.3. Reserved words

This topic describes all reserved words in SQL statements that are supported by Log Service.

```
AND
AS
BETWEEN
BY
CASE
CAST
CROSS
CUBE
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
DISTINCT
ELSE
END
ESCAPE
EXCEPT
EXISTS
FROM
GROUP
GROUPING
HAVING
IN
INNER
INSERT
INTERSECT
INTO
IS
JOIN
LEFT
LIKE
LIMIT
LOCALTIME
LOCALTIMESTAMP
NATURAL
NOT
NULL
ON
OR
ORDER
OUTER
RIGHT
ROLLUP
SELECT
THEN
TRUE
UNION
UNNEST
VALUES
WHEN
WHERE
WITH
```

8.4. Column aliases

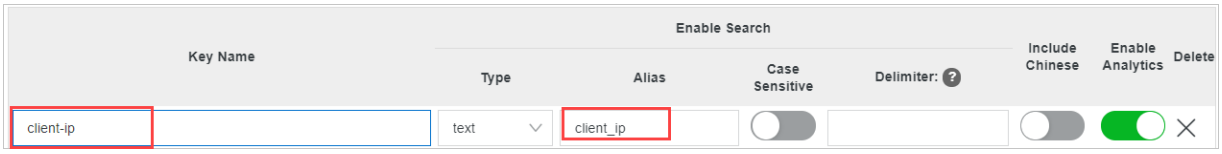
This topic describes the naming conventions of aliases. This topic also provides examples of aliases.

Naming conventions

A column name specified in an SQL statement can contain letters, digits, and underscores (_). The name must start with a letter. If you specify a column name that does not comply with the SQL-92 syntax when you collect logs, you must specify an alias for the column name when you configure indexes. For more information about how to configure indexes, see Configure indexes.

You can specify a short alias for a column whose original name is long and use the alias in an analytic statement.

Notice You can use aliases only in analytic statements. You must use original column names in search statements.



Examples

Original column name	Alias
User-Agent	User_Agent
User.Agent	user_agent
123	col
abceefghijklmnopqrstuvw	abc

8.5. Subqueries

A subquery is a query in which a SELECT statement is nested inside another SELECT statement. You can use subqueries to meet complex analysis requirements.

Syntax

Specify the FROM clause in a SELECT statement.

```
* | SELECT key FROM (sub_query)
```

Notice

- You must enclose the subquery statement in the FROM clause in parentheses `()`.
- If you want to analyze log data in the current Logstore, you must specify the keyword `FROM log`.

Examples

Example 1

Calculate the number of page views (PVs) by request method and obtain the minimum number of PVs.

- Query statement

```
* |
SELECT
 min(PV)
FROM  (
    SELECT
      count(1) as PV
    FROM      log
    GROUP BY
      request_method
  )
```

- Query and analysis results

min
13

Example 2

Calculate the ratio of the PVs in the current hour to the PVs in the same time period on the previous day. The time range for the query is **1 hour (on the hour)**. 86400 indicates the result of the current time minus 86400 seconds, which is equivalent to 1 day. log indicates the name of the Logstore.

- Query statement

```
* |
SELECT
  diff [1] AS today,
  diff [2] AS yesterday,
  diff [3] AS ratio
FROM  (
    SELECT
      compare(PV, 86400) AS diff
    FROM      (
        SELECT
          count(*) AS PV
        FROM          log
      )
  )
```

- Query and analysis results

today	yesterday	ratio
3337.0	3522.0	0.947473026689381

  - **3337.0** indicates the PVs in the current hour. Example: the PVs from 14:00:00 to 15:00:00 on December 25, 2020.
  - **3522.0** indicates the PVs in the same time period on the previous day. Example: the PVs from 14:00:00 to 15:00:00 on December 24, 2020.
  - **0.947473026689381** indicates the ratio of the PVs in the current hour to the PVs in the same time period on the previous day.

Example 3

Calculate the number of PVs on each page and the percentage of the PVs on each page to the total PVs.

- Query statement

```
* |
SELECT
  request_uri AS "Access page",
  c as "PVs",
  round(c * 100.0 /(sum(c) over()), 2) AS "Percentage%"
FROM  (
    SELECT
      request_uri AS request_uri,
      count(*) AS c
    FROM      log
    GROUP BY
      request_uri
    ORDER BY
      c DESC
  )
```

- Query and analysis results

Access page	PVs	Percentage%
/request/path-2/file-4	250	2.79
/request/path-3/file-6	244	2.72
/request/path-2/file-3	242	2.7
/request/path-1/file-5	237	2.65
/request/path-3/file-0	237	2.65
/request/path-3/file-2	236	2.64

8.6. Join queries on a Logstore and a MySQL database

Log Service allows you to use the JOIN syntax to query data from a Logstore and a MySQL database. The query results are saved to the database.

Prerequisites

An external store is created. For more information, see Associate Log Service with a MySQL database.

Procedure

1.
2.
3.
4. Execute a query statement.

   Log Service supports the following JOIN syntax:

   ```
   [ INNER ] JOIN
   LEFT [ OUTER ] JOIN
   RIGHT [ OUTER ] JOIN
   FULL [ OUTER ] JOIN
   ```

   The following sample code provides an example of a join query. For more information, see Associate a Logstore with a MySQL database to perform query and analysis.

   ```
   method:postlogstorelogs | select count(1) , histogram(logstore) from log  l join join_meta m on l
   .projectid = cast( m.ikey as varchar)
   ```

> **Notice**
>
> - You can use the JOIN syntax only on a Logstore and a small table in a MySQL database. A small table contains less than 20 MB of data.
> - In a query statement, the name of the Logstore must precede the join keyword, and the name of the external store must follow the join keyword.
> - You must specify the name of the external store in a query statement. When the system executes the statement, the system replaces the name with a combination of the database name and the table name. Do not enter only the table name.

5. Save the query results to the MySQL database.

   Log Service allows you to insert the query results into the database by using an INSERT statement. The following sample code provides an example of an INSERT statement:

```
method:postlogstorelogs | insert into method_output  select cast(method as varchar(65535)),count(
1) from log group by method
```

Sample Python script

```
# encoding: utf-8
from __future__ import print_function
from aliyun.log import *
from aliyun.log.util import base64_encodestring
from random import randint
import time
import os
from datetime import datetime
    endpoint = os.environ.get('ALIYUN_LOG_SAMPLE_ENDPOINT', 'cn-chengdu.log.aliyuncs.com')
    accessKeyId = os.environ.get('ALIYUN_LOG_SAMPLE_ACCESSID', '')
    accessKey = os.environ.get('ALIYUN_LOG_SAMPLE_ACCESSKEY', '')
    logstore = os.environ.get('ALIYUN_LOG_SAMPLE_LOGSTORE', '')
    project = "ali-yunlei-chengdu"
    client = LogClient(endpoint, accessKeyId, accessKey, token)
    # Create an external store.
    res = client.create_external_store(project,ExternalStoreConfig("rds_store","region","rds-vpc","vp
c id","Instance ID","Instance IP address","Instance port","Username","Password","Database name","Tabl
e name"));
    res.log_print()
    # Retrieve the details about the external store.
    res = client.get_external_store(project,"rds_store");
    res.log_print()
    res = client.list_external_store(project,"");
    res.log_print();
    # Execute a join query.
    req = GetLogsRequest(project,logstore,From,To,"","select count(1) from  "+ logstore +"  s join  m
eta m on  s.projectid = cast(m.ikey as varchar)");
    res = client.get_logs(req)
    res.log_print();
     # Save the query results to the MySQL database.
    req = GetLogsRequest(project,logstore,From,To,"," insert into rds_store select count(1) from  "+
logstore );
    res = client.get_logs(req)
    res.log_print();
```

9.Machine learning syntax and functions

9.1. Overview

Log Service provides the machine learning feature that supports multiple algorithms and calling methods. You can use the analytic statement and machine learning functions to call machine learning algorithms to analyze the characteristics of one or more fields within a period of time.

Log Service offers various time series analysis algorithms. You can call these algorithms to solve problems that are related to time series data. For example, you can predict time series, detect time series anomalies, decompose time series, and cluster multiple time series. In addition, the algorithms are compatible with standard SQL functions. This simplifies the usage of the algorithms and improves the efficiency of troubleshooting.

Features

- Supports various smooth operations on single-time series data.
- Supports algorithms that are used for the prediction, anomaly detection, change point detection, inflection point detection, and multi-period estimation of single-time series data.
- Supports decomposition operations on single-time series data.
- Supports various clustering algorithms of multi-time series data.
- Supports multi-field pattern mining (based on the sequence of numeric data or text).

Limits

When you use the machine learning feature of Log Service, you must take note of the following limits:

- The specified time series data must be sampled based on the same interval.
- The specified time series data cannot contain data that is repeatedly sampled from the same point in time.
- The processing capacity cannot exceed the maximum capacity. The following table describes the limits.

Item	Limit
Capacity of the time-series data processing	Data can be collected from a maximum of 150,000 consecutive points in time. If the data volume exceeds the processing capacity, you must aggregate the data or reduce the sampling amount.
Capacity of the density-based clustering algorithm	A maximum of 5,000 time series curves can be clustered at a time. Each curve cannot contain more than 1,440 points in time.
Capacity of the hierarchical clustering algorithm	A maximum of 2,000 time series curves can be clustered at a time. Each curve cannot contain more than 1,440 points in time.

Machine learning functions

Category		Function	Description
	Smooth function	ts_smooth_simple	Uses the Holt Winters algorithm to smooth time series data.
		ts_smooth_fir	Uses the finite impulse response (FIR) filter to smooth time series data.

Category		Function	Description
Time series		ts_smooth_iir	Uses the infinite impulse response (IIR) filter to smooth time series data.
	Multi-period estimation function	ts_period_detect	Estimates time series data by period.
	Change point detection function	ts_cp_detect	Detects the intervals in which data has different statistical features. The interval endpoints are change points.
		ts_breakout_detect	Detects the points in time at which data experiences dramatic changes.
	Maximum value detection function	ts_find_peaks	Detects the local maximum value of time series data in a specified window.
	Prediction and anomaly detection function	ts_predicate_simple	Uses default parameters to model time series data, predict time series data, and detect anomalies.
		ts_predicate_ar	Uses an autoregressive (AR) model to model time series data, predict time series data, and detect anomalies.
		ts_predicate_arma	Uses an autoregressive moving average (ARMA) model to model time series data, predict time series data, and detect anomalies.
		ts_predicate_arima	Uses an autoregressive integrated moving average (ARIMA) model to model time series data, predict time series data, and detect anomalies.
		ts_regression_predict	Predicts the long-run trend for a single periodic time series.
	Sequence decomposition function	ts_decompose	Uses the Seasonal and Trend decomposition using Loess (STL) algorithm to decompose time series data.
	Time series clustering function	ts_density_cluster	Uses a density-based clustering method to cluster multiple time series.
		ts_hierarchical_cluster	Uses a hierarchical clustering method to cluster multiple time series.
		ts_similar_instance	Queries time series curves that are similar to a specified time series curve.

Category		Function	Description
	Kernal density estimation functions	kernel_density_estimation	Uses the smooth peak function to fit the observed data points. In this way, the function simulates the real probability distribution curve.
	Time series padding function	series_padding	Pads data points that are missing in a time series.
	Anomaly comparison function	anomaly_compare	Compares the degree of difference of an observed object in two periods of time.
Pattern mining	Frequent pattern statistical function	pattern_stat	Mines representative combinations of attributes among the given multi-attribute field samples to obtain the frequent pattern in statistical patterns.
	Differential pattern statistical function	pattern_diff	Identifies the pattern that causes differences between two collections in specified conditions.
	Root cause analysis function	rca_kpi_search	Analyze the subdimension attributes that cause anomalies of the monitoring metric.
	Correlation analysis functions	ts_association_analysis	Identifies the metrics that are correlated to a specified metric among multiple observed metrics in the system.
		ts_similar	Identifies the metrics that are correlated to specified time series data among multiple observed metrics in the system.
	Request URL classification function	url_classify	Classifies a request URL and attaches a tag to the URL. The function also provides the regular expression that defines the pattern of the tag.

9.2. Smooth functions

This topic describes the smooth functions that you can use to smooth and filter specified time series curves. Filtering is the first step to discover the shape of time series curves.

Function list

Function	Description
ts_smooth_simple	Uses the Holt-Winters forecasting algorithm to filter time series data. This function is the default smooth function.
ts_smooth_fir	Filters time series data by using a finite impulse response (FIR) filter.

Function	Description
ts_smooth_iir	Filter time series data using an infinite impulse response (IIR) filter.

ts_smooth_simple

- Function format:

```
select ts_smooth_simple(x, y)
```

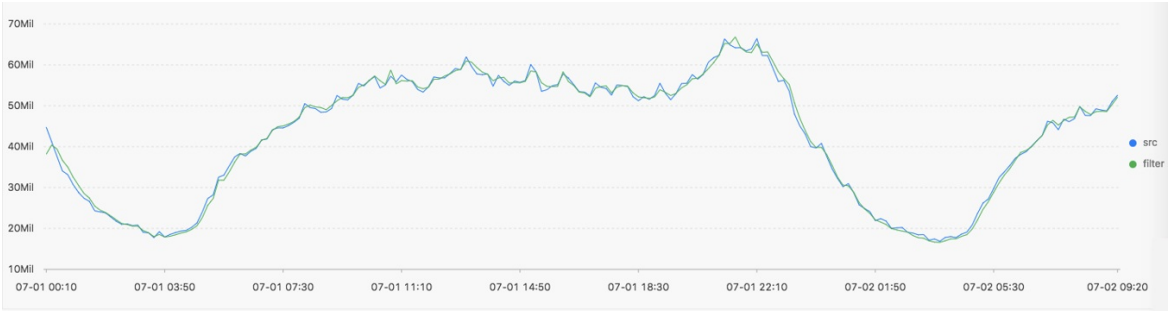
- The following table lists the parameters of the function format.

Parameter	Description	Value
x	The time sequence. Points in time are sorted in ascending order along the horizontal axis.	Each point in time is a Unix timestamp. Unit: seconds.
y	The sequence of numeric data corresponding to each specified point in time.	N/A.

- Examples
  - The query statement is as follows:

```
* | select ts_smooth_simple(stamp, value) from ( select __time__ - __time__ % 120 as stamp, avg(v) as value from log GROUP BY stamp order by stamp )
```

  - Output result



- The following table lists the display items.

Display item		Description
Horizontal axis	unixtime	Each point in time is a Unix timestamp. Unit: seconds.
Vertical axis	src	The raw data.
	filter	The data generated after the filtering operation is performed.

ts_smooth_fir

- Function format:

- If you cannot determine filter parameters, use the built-in window parameters in the following statement:

```
select ts_smooth_fir(x, y,winType,winSize)
```

- If you can determine filter parameters, you can set the parameters as needed in the following statement:

```
select ts_smooth_fir(x, y,array[])
```

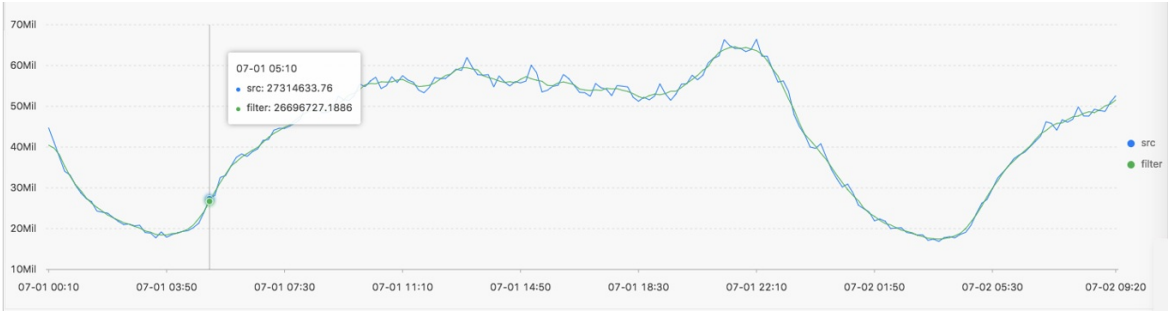
- The following table lists the parameters of the function format.

Parameter	Description	Value
x	The time sequence. Points in time are sorted in ascending order along the horizontal axis.	Each point in time is a Unix timestamp. Unit: seconds.
y	The sequence of numeric data corresponding to each specified point in time.	N/A.
winType	The type of window for filtering.	Valid values: ○ rectangle: a rectangular window ○ hanning: a Hanning window ○ hamming: a Hamming window ○ blackman: a Blackman window **Note** We recommend that you set this parameter to rectangle for better display.
winSize	The length of the filter window.	The value is of the long data type. Valid values: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, and 15.
array[]	Used to calculate coefficients for the FIR filter.	The value is an array where the sum of elements is 1. Example: array[0.2, 0.4, 0.3, 0.1].

- Example 1
  - The query statement is as follows:

```
* | select ts_smooth_fir(stamp, value, 'rectangle', 4) from ( select __time__ - __time__ % 120 a
s stamp, avg(v) as value from log GROUP BY stamp order by stamp )
```

  - Output result

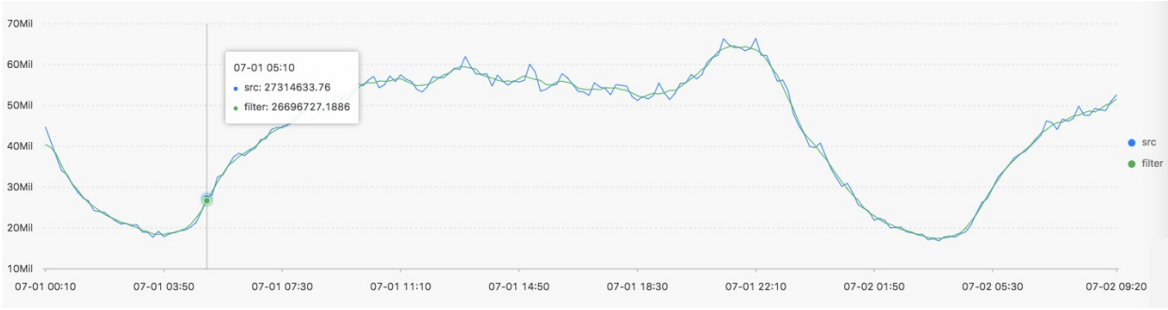


- Example 2

- The query statement is as follows:

```
* | select ts_smooth_fir(stamp, value, array[0.2, 0.4, 0.3, 0.1]) from ( select __time__ - __time__ % 120 as stamp, avg(v) as value from log GROUP BY stamp order by stamp )
```

- Output result



- The following table lists the display items.

Display item		Description
Horizontal axis	unixtime	Each point in time is a Unix timestamp. Unit: seconds.
Vertical axis	src	The raw data.
	filter	The data generated after the filtering operation is performed.

ts_smooth_iir

- Function format:

```
select ts_smooth_iir(x, y, array[], array[] )
```

- The following table lists the parameters of the function format.

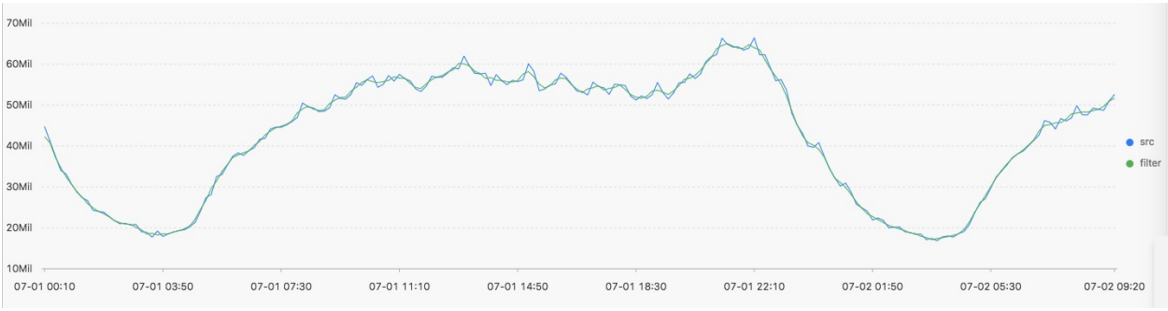
Parameter	Description	Value
x	The time sequence. Points in time are sorted in ascending order along the horizontal axis.	Each point in time is a Unix timestamp. Unit: seconds.
y	The sequence of numeric data corresponding to each specified point in time.	N/A.
array[]	Used to calculate the filter coefficients related to x_i for the IIR filter.	The value is an array where the sum of elements is 1. Valid lengths of elements: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, and 15. Example: array[0.2, 0.4, 0.3, 0.1].
array[]	The type of the filter that specifies the algorithm to compute the filter coefficients related to y_{i-1} for the IIR filter.	The value is an array where the sum of elements is 1. Valid lengths of elements: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, and 15. Example: array[0.2, 0.4, 0.3, 0.1].

- Examples

- The query statement is as follows:

```
* | select ts_smooth_iir(stamp, value, array[0.2, 0.4, 0.3, 0.1], array[0.4, 0.3, 0.3]) from ( select __time__ - __time__ % 120 as stamp, avg(v) as value from log GROUP BY stamp order by stamp )
```

- Output result



- The following table lists the display items.

Display item		Description
Horizontal axis	unixtime	Each point in time is a Unix timestamp. Unit: seconds.
Vertical axis	src	The raw data.
	filter	The data generated after the filtering operation is performed.

9.3. Multi-period estimation functions

This topic describes multi-period estimation functions that you can use to estimate the periodicity of time series data distributed in different time intervals. This topic also describes how to extract the periodicity by using a series of operations such as Fourier transform (FT).

Function list

Function	Description
ts_period_detect	Estimates the periodicity of time series data distributed in different time intervals.
ts_period_classify	Uses FT to calculate the periodicity of specified time series curves. This function can be used to identify periodic curves.

ts_period_detect

Function format:

```
select ts_period_detect(x,y,minPeriod,maxPeriod)
```

The following table lists the parameters in the function.

Parameter	Description	Value

Parameter	Description	Value
x	The time sequence. Points in time are sorted in ascending order along the horizontal axis.	Each point in time is a UNIX timestamp. Unit: seconds.
y	The sequence of numeric data at a specific point in time.	None.
minPeriod	The ratio of the minimum length of a time series data within a period to the total length of the time series data. The ratio is estimated based on your time series curve.	The parameter value must be a decimal number. Valid values: (0.0, 1.0].
maxPeriod	The ratio of the maximum length of a time series within a period to the total length of the time series data. The ratio is estimated based on your time series curve. **Notice** The value of the *maxPeriod* parameter must be greater than that of the *minPeriod* parameter. The value must be less than 0.5. If you set the *maxPeriod* parameter to a value greater than 0.5, the system automatically changes the value to 0.5.	The parameter value must be a decimal number. Valid values: (0.0, 1.0].

Example

- The following query statement is executed:

```
* | select ts_period_detect(stamp, value, 0.2, 0.5) from ( select __time__ - __time__ % 120 as sta
mp, avg(v) as value from log GROUP BY stamp order by stamp )
```

- Output result

  The output result is of the array type. The result contains UNIX timestamps, statistical values (such as average traffic), and status codes. Each red circle in the following figure represents a status code whose value is 1.0. The following figure shows the output result.

  Each shaded part between two consecutive red circles in the following figure represents a period. The curve of each period tends to be the same.

ts_period_classify

Function format:

```
select ts_period_classify(stamp,value,instanceName)
```

The following table lists the parameters in the function.

Parameter	Description	Value
stamp	The time sequence. Points in time are sorted in ascending order along the horizontal axis.	Each point in time is a UINX timestamp. Unit: seconds.
value	The sequence of numeric data at a specific point in time.	None.
instanceName	The name of the time series curve.	None.

Example:

- The following query statement is executed:

```
* and h : nu2h05202.nu8 | select ts_period_classify(stamp, value, name) from log
```

- Response

line_name	prob	type
asg-2zxjqn6zf5ewg188pg5	1.0	-1.0
asg-bp1j8snc92p6v5pptgpj	0.07203669207039314	0.0
asg-wz99hse7u4ubopo5dt9o	0.0	0.0
asg-bp18oqni0gq96vy85te4	0.05590892692207093	0.0

The following table lists the display items.

Display item	Description
line_name	The name of the time series curve.
prob	The ratio of the number of values within the primary period to the total number of values on the time series curve. Valid values: [0, 1]. You can set the value to 0.15 for testing.
type	The type of the time series curve. Valid values: -1, -2, and 0. • The value -1 indicates that the length of the time series curve is too short (less than 64 points). • The value -2 indicates that the time series curve has a high failure rate (higher than 20%). • The value 0 indicates that the time series curve is periodic.

9.4. Change point detection function

The change point detection function detects change points in time series data.

The change point detection function supports two types of change points:

- Changes of statistical characteristics within a specified time period
- Obvious faulting in a sequence

Function list

Function	Description
ts_cp_detect	This function finds intervals with different statistical characteristics within a time series. The interval endpoints are change points.
ts_breakout_detect	This function finds the time point when statistics steeply increase or decrease within a time series.

ts_cp_detect

Function format:

- If you are not sure about the window size, use the ts_cp_detect function in the following format. Then, the algorithm called by the function will use a window with a length of 10 to detect change points.

```
select ts_cp_detect(x, y, amplePeriod,sampleMethod)
```

- If you need to adjust the display effect of a service curve, use the ts_cp_detect function in the following format. Then, you can optimize the display effect by setting the minSize parameter.

```
select ts_cp_detect(x, y, minSize, samplePeriod, sampleMethod)
```

The following table describes the parameters.

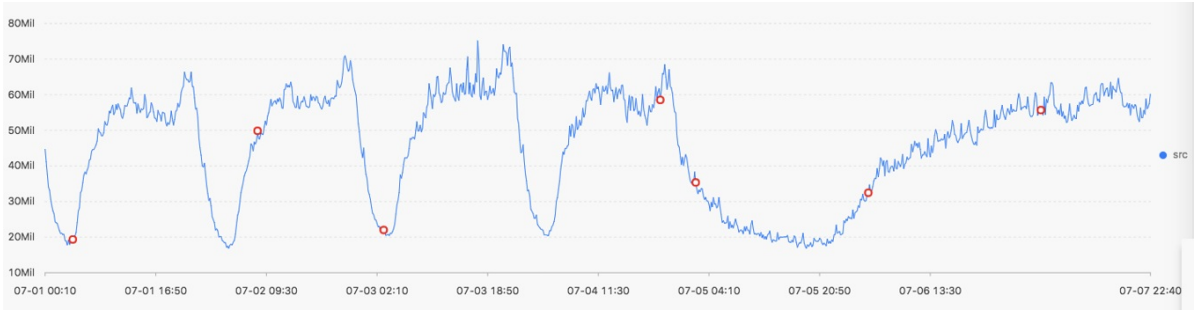
Parameter	Description	Value
x	Time column in ascending order	Unixtime timestamp in seconds
y	Numeric column corresponding to the data at a specified time point	-
minSize	Minimum length of consecutive intervals	The minimum value is 3, and the maximum value cannot exceed 1/10 of the length of the current input data.
samplePeriod	Period during which the current time series data is sampled	Long type values ranging from 1 to 86399 seconds
sampleMethod	Method for sampling the data in the sampling window	Value range: • avg: average value of the data in the window • max: maximum value of the data in the window • min: minimum value of the data in the window • sum: sum of the data in the window

Example:

- Statement for query and analysis:

```
* | select ts_cp_detect(stamp, value, 3, 1, 'avg') from (select __time__ - __time__ % 10 as stamp,
avg(v) as value from log GROUP BY stamp order by stamp)
```

- Result:



The following table describes the display items.

Display item		Description
Horizontal axis	unixtime	Data timestamp in seconds, for example, 1537071480
Longitudinal axis	src	Data before filtering, for example, 1956092.7647745228
	prob	Probability that a point is a change point. Its value ranges from 0 to 1.

ts_breakout_detect

Function format:

```
select ts_breakout_detect(x, y, winSize, samplePeriod, sampleMethod)
```

The following table describes the parameters.

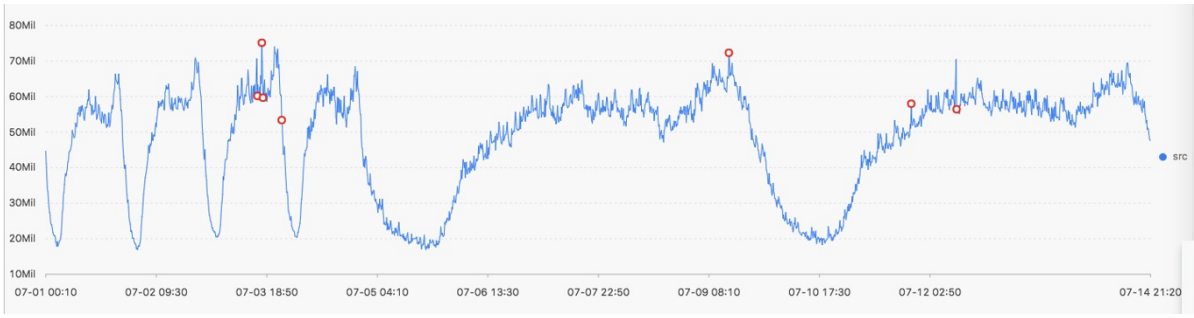
Parameter	Description	Value
x	Time column in ascending order	Unixtime timestamp in seconds
y	Numeric column corresponding to the data at a specified time point	-
winSize	Minimum length of consecutive intervals	The minimum value is 3, and the maximum value cannot exceed 1/10 of the length of the current input data.
samplePeriod	Period during which the current time series data is sampled	Long type values ranging from 1 to 86399 seconds
sampleMethod	Method for sampling the data in the sampling window	Value range: • avg: average value of the data in the window • max: maximum value of the data in the window • min: minimum value of the data in the window • sum: sum of the data in the window

Example:

- Statement for query and analysis:

```
* | select ts_breakout_detect(stamp, value, 3, 1, 'avg') from (select __time__ - __time__ % 10 as stamp, avg(v) as value from log GROUP BY stamp order by stamp)
```

- Result:



The following table describes the display items.

Display item		Description
Horizontal axis	unixtime	Data timestamp in seconds, for example, 1537071480
Longitudinal axis	src	Data before filtering, for example, 1956092.7647745228
	prob	Probability that a point is a change point. Its value ranges from 0 to 1.

9.5. Maximum value detection functions

This topic describes how to use maximum value detection functions to find the locally maximum value of time series data in a specified window.

ts_find_peaks

Function format:

```
select ts_find_peaks(x, y, winSize)
```

The following table lists the parameters of the function format.

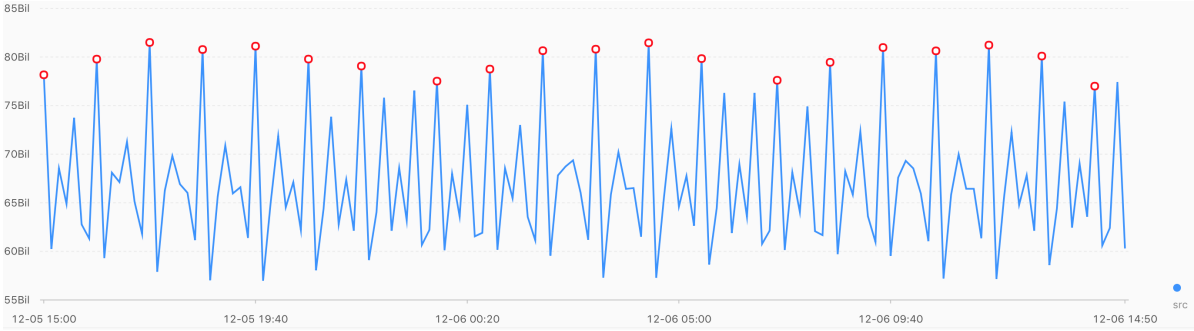
Parameter	Description	Value
x	The time sequence. Points in time are sorted in ascending order along the horizontal axis.	Each point in time is a Unix timestamp. Unit: seconds.
y	The sequence of numeric data corresponding to each specified point in time.	N/A.
winSize	The minimum length of the detection window.	The value of the parameter is of the long data type, ranging from 1 to the length of time series data. We recommend that you set this parameter to one tenth of the actual data length.

Example:

- The query statement is as follows:

```
* and h : nu2h05202.nu8 and m: NET |  select ts_find_peaks(stamp, value, 30) from (select __time__
- __time__ % 10 as stamp, avg(v) as value from log GROUP  BY  stamp order by stamp)
```

- The following figure shows the output result.



The following table lists the display items.

Display item		Description
Horizontal axis	unixtime	The timestamp of time series data. Unit: seconds. Example: 1537071480.
Vertical axis	src	The raw data. Example: 1956092.7647745228.
	peak_flag	Indicates whether the numeric value at the time point is the maximum value. Valid values: • 1.0: The numeric value at the time point is the maximum value. • 0.0: The numeric value at the time point is not the maximum value.

9.6. Prediction and anomaly detection functions

To detect anomalies, you can use a prediction and anomaly detection function to predict a time series curve as well as identify the Ksigma and quantiles of the errors between a predicted curve and an actual curve.

Function list

Function	Description
ts_predicate_simple	Uses default parameters to model time series data and performs simple time series prediction and anomaly detection.
ts_predicate_ar	Uses an autoregressive model (AR) model to model time series data and performs simple time series prediction and anomaly detection.
ts_predicate_arma	Uses an autoregressive moving average (ARMA) model to model time series data and performs simple time series prediction and anomaly detection.
ts_predicate_arima	Uses an autoregressive integrated moving average (ARIMA) model to model time series data and performs simple time series prediction and anomaly detection.

Function	Description
ts_regression_predict	Accurately predicts the trend for a periodic time series curve. Scenario: This function can be used to predict metering data, network traffic, financial data, and different business data that follows certain rules.
ts_anomaly_filter	Filters the anomalies detected during anomaly detection on multiple time series curves based on the custom anomaly mode. This function helps you quickly find abnormal curves.

ts_predicate_simple

Function format:

```
select ts_predicate_simple(x, y, nPred, isSmooth)
```

The following table lists the parameters of the function format.

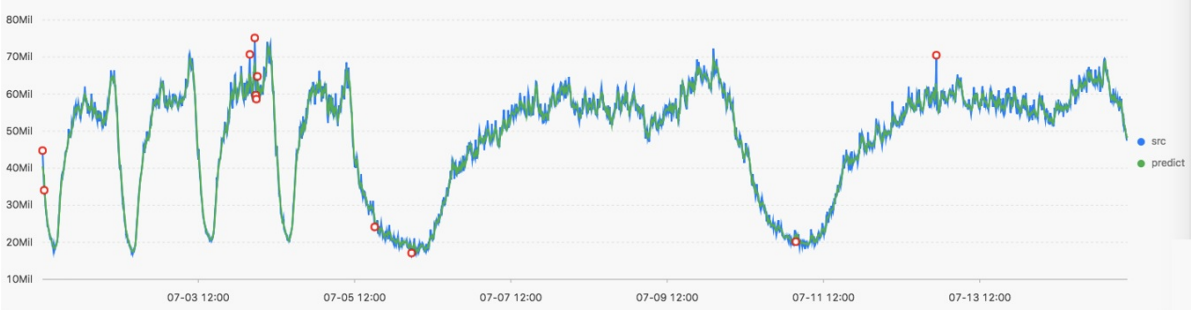
Parameter	Description	Value
x	The time sequence. Points in time are sorted in ascending order along the horizontal axis.	Each point in time is a Unix timestamp. Unit: seconds.
y	The sequence of numeric data corresponding to each specified point in time.	N/A.
nPred	The number of points for prediction.	The value is of the long data type and must be equal to or greater than 1.
isSmooth	Specifies whether to filter the raw data.	The value is of the Boolean data type. The default value is true, which indicates that the raw data is to filter.

Example:

- The query statement is as follows:

```
* | select ts_predicate_simple(stamp, value, 6) from (select __time__ - __time__ % 60 as stamp, av
g(v) as value from log GROUP BY stamp order by stamp)
```

- Output result



The following table lists the display items.

Display item		Description
Horizontal axis	unixtime	The Unix timestamp of the data. Unit: seconds.

Display item		Description
Vertical axis	src	The raw data.
	predict	The data generated after the filtering operation is performed.
	upper	The upper limit of the confidence interval. The confidence level is 0.85, which cannot be modified.
	lower	The lower limit of the confidence interval. The confidence level is 0.85, which cannot be modified.
	anomaly_prob	The probability that the point is an anomaly. Valid values: [0, 1].

ts_predicate_ar

Function format:

```
select ts_predicate_ar(x, y, p, nPred, isSmooth)
```

The following table lists the parameters of the function format.

Parameter	Description	Value
x	The time sequence. Points in time are sorted in ascending order along the horizontal axis.	Each point in time is a Unix timestamp. Unit: seconds.
y	The sequence of numeric data corresponding to each specified point in time.	N/A.
p	The order of the AR model.	The value is of the long data type. Valid values: 2, 3, 4, 5, 6, 7, and 8.
nPred	The number of points for prediction.	The value is of the long data type. Valid values: [1, 5 × *p*].
isSmooth	Specifies whether to filter the raw data.	The value is of the Boolean data type. The default value is true, which indicates that the raw data is to filter.

An example of the query statement is as follows:

```
* | select ts_predicate_ar(stamp, value, 3, 4) from (select __time__ - __time__ % 60 as stamp, avg(v) as value from log GROUP BY stamp order by stamp)
```

Note The output result is similar to that of the ts_predicate_simple function. For more information, see the output result of the ts_predicate_simple function.

ts_predicate_arma

Function format:

```
select ts_predicate_arma(x, y, p, q, nPred, isSmooth)
```

The following table lists the parameters of the function format.

Parameter	Description	Value
x	The time sequence. Points in time are sorted in ascending order along the horizontal axis.	Each point in time is a Unix timestamp. Unit: seconds.
y	The sequence of numeric data corresponding to each specified point in time.	N/A.
p	The order of the AR model.	The value is of the long data type. Valid values: [2, 100].
q	The order of the ARMA model.	The value is of the long data type. Valid values: 2, 3, 4, 5, 6, 7, and 8.
nPred	The number of points for prediction.	The value is of the long data type. Valid values: [1, 5 × *p*].
isSmooth	Specifies whether to filter the raw data.	The value is of the Boolean data type. The default value is true, which indicates that the raw data is to filter.

An example of the query statement is as follows:

```
* | select ts_predicate_arma(stamp, value, 3, 2, 4) from (select __time__ - __time__ % 60 as stamp, avg(v) as value from log GROUP BY stamp order by stamp)
```

Note The output result is similar to that of the ts_predicate_simple function. For more information, see the output result of the ts_predicate_simple function.

ts_predicate_arima

Function format:

```
select ts_predicate_arima(x, y, p, d, q, nPred, isSmooth)
```

The following table lists the parameters of the function format.

Parameter	Description	Value
x	The time sequence. Points in time are sorted in ascending order along the horizontal axis.	Each point in time is a Unix timestamp. Unit: seconds.
y	The sequence of numeric data corresponding to each specified point in time.	N/A.
p	The order of the AR model.	The value is of the long data type. Valid values: 2, 3, 4, 5, 6, 7, and 8.
d	The order of the ARIMA model.	The value is of the long data type. Valid values: [1, 3].
q	The order of the ARMA model.	The value is of the long data type. Valid values: 2, 3, 4, 5, 6, 7, and 8.

Parameter	Description	Value
nPred	The number of points for prediction.	The value is of the long data type. Valid values: [1, 5 × *p*].
isSmooth	Specifies whether to filter the raw data.	The value is of the Boolean data type. The default value is true, which indicates that the raw data is to filter.

An example of the query statement is as follows:

```
* | select ts_predicate_arima(stamp, value, 3, 1, 2, 4, 1, 'avg') from (select __time__ - __time__ % 60 as stamp, avg(v) as value from log GROUP BY stamp order by stamp)
```

Note The output result is similar to that of the ts_predicate_simple function. For more information, see the output result of the ts_predicate_simple function.

ts_regression_predict

Function format:

```
select ts_regression_predict(x, y, nPred, algotype,processType)
```

The following table lists the parameters of the function format.

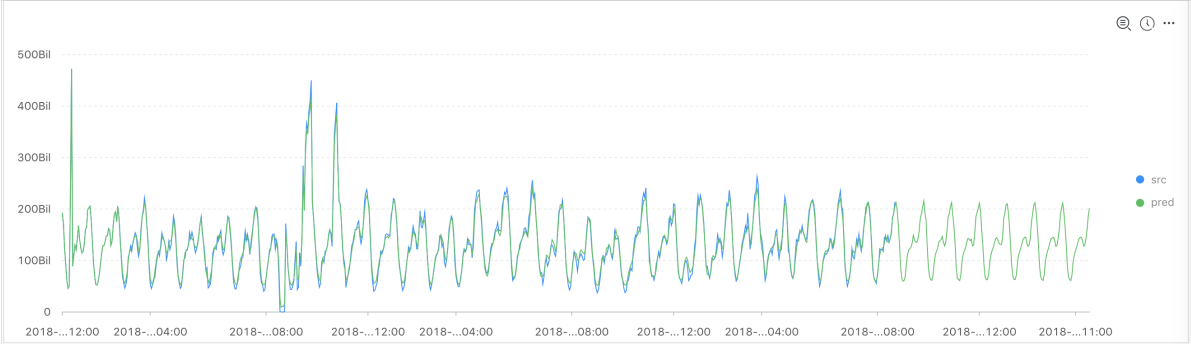
Parameter	Description	Value
x	The time sequence. Points in time are sorted in ascending order along the horizontal axis.	Each point in time is a Unix timestamp. Unit: seconds.
y	The sequence of numeric data corresponding to each specified point in time.	N/A.
nPred	The number of points for prediction.	The value is of the long data type. Valid values: [1, 500].
algotype	The algorithm type for prediction.	Valid values: • origin: uses the Gradient Boosted Regression Tree (GBRT) algorithm for prediction. • forest: uses the GBRT algorithm for prediction based on the trend components decomposed by Seasonal and Trend decomposition using Loess (STL), and then uses the additive model to sum up the decomposed components and obtains the predicted data. • linear: uses the Linear Regression algorithm for prediction based on the trend components decomposed by STL, and then uses the additive model to sum up the decomposed components and obtains the predicted data.
processType	Specifies whether to preprocess the data.	Valid values: • 0: No additional data preprocessing is performed. • 1: Abnormal data is removed before prediction.

Example:

- The query statement is as follows:

```
* and h : nu2h05202.nu8 and m: NET |  select ts_regression_predict(stamp, value, 200, 'origin') from (select __time__ - __time__ % 60 as stamp, avg(v) as value from log GROUP  BY  stamp order by stamp)
```

- Output result



The following table lists the display items.

Display item		Description
Horizontal axis	unixtime	The Unix timestamp of the data. Unit: seconds.
Vertical axis	src	The raw data.
	predict	The data generated after the filtering operation is performed.

ts_anomaly_filter

Function format:

```
select ts_anomaly_filter(lineName, ts, ds, preds, probs, nWatch, anomalyType)
```

The following table lists the parameters of the function format.

Parameter	Description	Value
lineName	The name of each curve. The value is of the varchar type.	N/A
ts	The time sequence of the curve, which indicates the time on the current curve. The parameter value is an array of points in time of the double data type sorted in ascending order.	N/A
ds	The actual value sequence of the curve. The parameter value is an array of data points of the double data type. This parameter value has the same length as the ts parameter value.	N/A

Parameter	Description	Value
preds	The predicted value sequence of the curve. The parameter value is an array of data points of the double data type. This parameter value has the same length as the ts parameter value.	N/A
probs	The sequence of anomaly detection results of the curve. The parameter value is an array of data points of the double data type. This parameter value has the same length as the ts parameter value.	N/A
nWatch	The number of the recently observed actual values on the curve. The value is of the long data type. The value must be smaller than the number of points in time on the curve.	N/A
anomalyType	The type of the anomaly to filter. The value is of the long data type.	Valid values: • 0: all anomalies • 1: positive anomalies • -1: negative anomalies

Example:

- The query statement is as follows:

```
* | select res.name, res.ts, res.ds, res.preds, res.probs
     from (
         select ts_anomaly_filter(name, ts, ds, preds, probs, cast(5 as bigint), cast(1 as bigint)
) as res
       from (
         select name, res[1] as ts, res[2] as ds, res[3] as preds, res[4] as uppers, res[5] as low
ers, res[6] as probs
     from (
         select name, array_transpose(ts_predicate_ar(stamp, value, 10)) as res
         from (
           select name, stamp, value from log where name like '%asg-%') group by name)) );
```

- Output result

```
| name                     | ts                                                          | ds          |
preds     | probs       |
| ------------------------ | ----------------------------------------------------------- | ----------- |
--------- | ----------- |
| asg-bp1hylzdi2wx7civ0ivk | [1.5513696E9, 1.5513732E9, 1.5513768E9, 1.5513804E9] | [1,2,3,NaN] |
[1,2,3,4] | [0,0,1,NaN] |
```

9.7. Sequence decomposition function

The sequence decomposition function can decompose service curves and highlight information about the curve trends and periods.

ts_decompose

Function format:

```
select ts_decompose(x, y, samplePeriod, sampleMethod)
```

The following table describes the parameters.

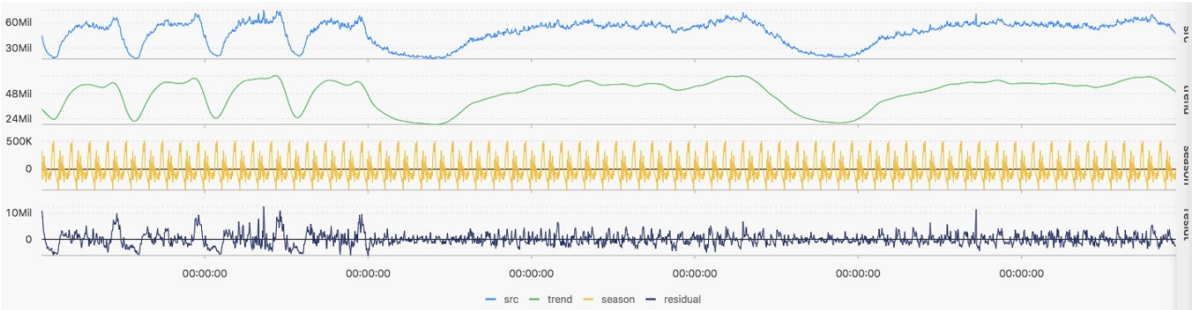
Parameter	Description	Value
x	Time column in ascending order	Unixtime timestamp in seconds
y	Numeric column corresponding to the data at a specified time point	-
samplePeriod	Period during which the current time series data is sampled	Long type values ranging from 1 to 86399 seconds
sampleMethod	Method for sampling the data in the sampling window	Value range: • avg: average value of the data in the window • max: maximum value of the data in the window • min: minimum value of the data in the window • sum: sum of the data in the window

Example:

- Statement for query and analysis:

```
* | select ts_decompose(stamp, value, 1, 'avg') from (select __time__ - __time__ % 60 as stamp, av
g(v) as value from log GROUP BY stamp order by stamp)
```

- Result:



The following table describes the display items.

Display item		Description
Horizontal axis	unixtime	Unixtime timestamp in seconds
Longitudinal axis	src	Raw data
	trend	Curve trend after decomposition
	season	Curve period after decomposition
	residual	Residual data after decomposition

9.8. Time series clustering functions

You can use a time series clustering function to cluster multiple pieces of time series data and obtain different curve shapes. Then, you can quickly find the corresponding cluster center and curves with shapes that are different from the curve shapes in the cluster.

Function list

Function	Description
ts_density_cluster	Uses a density-based clustering method to cluster multiple pieces of time series data.
ts_hierarchical_cluster	Uses a hierarchical clustering method to cluster multiple pieces of time series data.
ts_similar_instance	Queries curves that are similar to a specified curve.

ts_density_cluster

Function format:

```
select ts_density_cluster(x, y, z)
```

The following table describes the parameters.

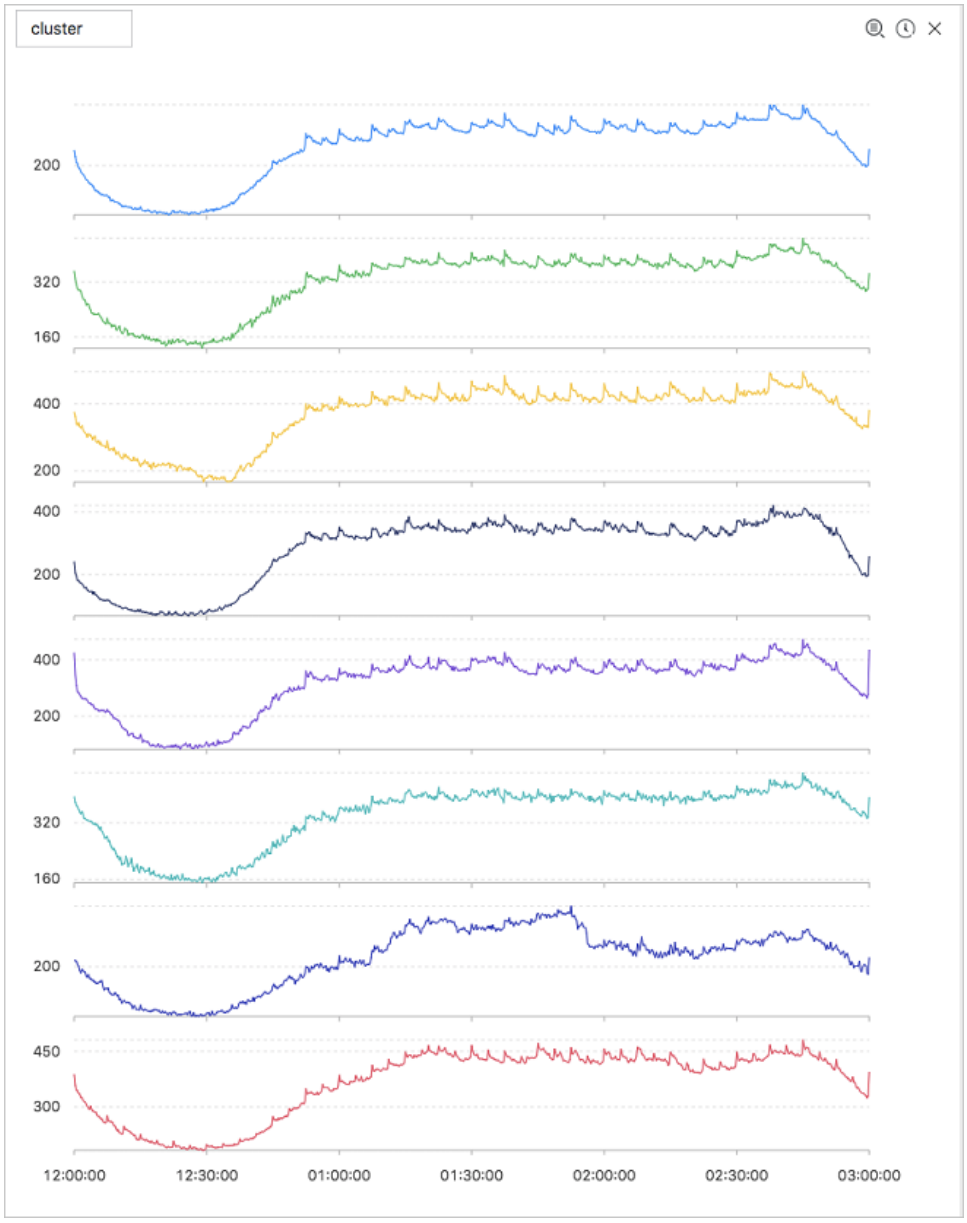
Parameter	Description	Value
x	The sequence of time in ascending order.	Unix timestamp. Unit: seconds.
y	The sequence of numeric data corresponding to each specified time point.	N/A
z	The metric name corresponding to the data at each specified time point.	String type, for example, machine01.cpu_usr.

Example:

- The statement for query and analysis is as follows:

```
* and (h: "machine_01" OR h: "machine_02" OR h : "machine_03") | select ts_density_cluster(stamp, metric_value, metric_name) from (select __time__ - __time__ % 600 as stamp, avg(v) as metric_value , h as metric_name from log GROUP BY stamp, metric_name order BY metric_name, stamp)
```

- The following figure shows the output result.



The following table describes the display items.

Display item	Description
cluster_id	The category of the cluster. The value -1 indicates that the cluster is not categorized in any cluster centers.
rate	The proportion of instances in the cluster.
time_series	The timestamp sequence of the cluster center.
data_series	The data sequence of the cluster center.
instance_names	The collection of instances included in the cluster center.
sim_instance	The name of an instance in the cluster.

ts_hierarchical_cluster

Function format:

```
select ts_hierarchical_cluster(x, y, z)
```

The following table describes the parameters.

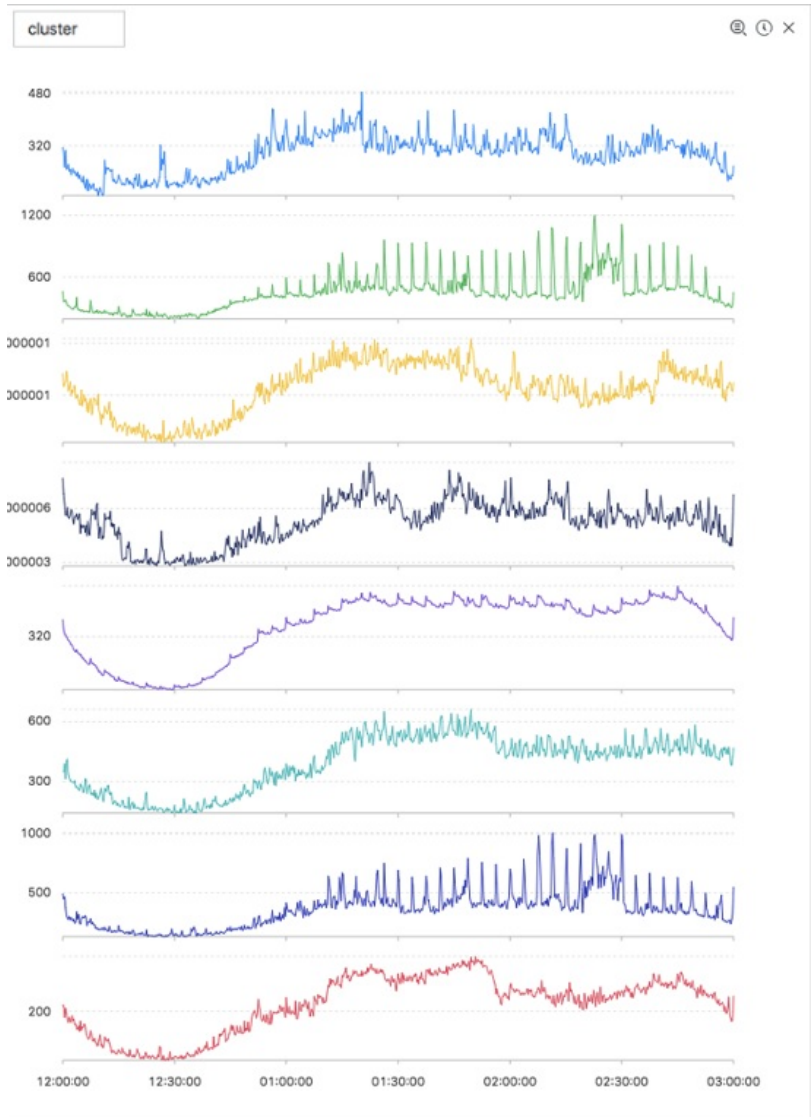
Parameter	Description	Value
x	The sequence of time in ascending order.	Unix timestamp. Unit: seconds.
y	The sequence of numeric data corresponding to each specified time point.	N/A
z	The metric name corresponding to the data at each specified time point.	String type, for example, machine01.cpu_usr.

Example:

- The statement for query and analysis is as follows:

```
* and (h: "machine_01" OR h: "machine_02" OR h : "machine_03") | select ts_hierarchical_cluster(stamp, metric_value, metric_name) from (select __time__ - __time__ % 600 as stamp, avg(v) as metric_value, h as metric_name from log GROUP BY stamp, metric_name order BY metric_name, stamp)
```

- The following figure shows the output result.



The following table describes the display items.

Display item	Description
cluster_id	The category of the cluster. The value -1 indicates that the cluster is not categorized in any cluster centers.
rate	The proportion of instances in the cluster.
time_series	The timestamp sequence of the cluster center.
data_series	The data sequence of the cluster center.
instance_names	The collection of instances included in the cluster center.
sim_instance	The name of an instance in the cluster.

ts_similar_instance

Function format:

```
select ts_similar_instance(x, y, z, instance_name, topK, metricType)
```

The following table describes the parameters.

Parameter	Description	Value
x	The sequence of time in ascending order.	Unix timestamp. Unit: seconds.
y	The sequence of numeric data corresponding to each specified time point.	N/A
z	The metric name corresponding to the data at each specified time point.	String type, for example, machine01.cpu_usr.
instance_name	The name of the specified metric to be queried in the z collection.	String type, for example, machine01.cpu_usr. **Note** The metric must be an existing one.
topK	The curves similar to a given curve. A maximum of K curves are returned.	N/A
metricType	The metric used to measure the similarity between time series curves.	{'shape', 'manhattan', 'euclidean'}

For example, the statement for query and analysis is as follows:

```
* and m: NET and m: Tcp and (h: "nu4e01524.nu8" OR  h: "nu2i10267.nu8" OR  h : "nu4q10466.nu8") | sel
ect ts_similar_instance(stamp, metric_value, metric_name, 'nu4e01524.nu8') from (select __time__ - __
time__ % 600 as stamp, sum(v) as metric_value, h as metric_name from log GROUP BY stamp, metric_name
order BY  metric_name, stamp)
```

The following table describes the display items.

Display item	Description
instance_name	The list of metrics that are similar to the specified metric.
time_series	The timestamp sequence of the cluster center.
data_series	The data sequence of the cluster center.

9.9. Frequent pattern statistical function

The frequent pattern statistical function mines representative combinations of attributes from the given multi-attribute field samples to summarize the current logs.

pattern_stat

Function format:

```
select pattern_stat(array[col1, col2, col3], array['col1_name', 'col2_name', 'col3_name'], array[col5
, col6], array['col5_name', 'col6_name'], supportScore, sample_ratio)
```

The following table describes the parameters.

Parameter	Description	Value
array[col1, col2, col3]	Input column composed of character type values	Values in array format, for example, array[clientIP, sourceIP, path, logstore]
array['col1_name', 'col2_name', 'col3_name']	Name corresponding to the input column composed of character type values	Values in array format, for example, array['clientIP', 'sourceIP', 'path', 'logstore']
array[col5, col6]	Input column composed of numeric values	Values in array format, for example, array[Inflow, OutFlow]
array['col5_name', 'col6_name']	Name corresponding to the input column composed of numeric values	Values in array format, for example, array['Inflow', 'OutFlow']
supportScore	Support level of positive and negative samples for pattern mining	Double type values. Range: (0,1].
sample_ratio	Sampling ratio with the default value of 0.1, which indicates that only 10% of the total samples are used	Double type values. Range: (0,1].

Example:

- Statement for query and analysis:

```
* | select pattern_stat(array[ Category, ClientIP, ProjectName, LogStore, Method, Source, UserAgent ], array[ 'Category', 'ClientIP', 'ProjectName', 'LogStore', 'Method', 'Source', 'UserAgent' ], array[ InFlow, OutFlow ], array[ 'InFlow', 'OutFlow' ], 0.45, 0.3) limit 1000
```

- Result:

count	supportscore	pattern
468235	0.9880626809484018	InFlow >= 0.0 and InFlow <= 60968.7 and OutFlow >= 0.0 and OutFlow <= 15566.4
459356	0.9693263443991458	Status = '200' and OutFlow >= 0.0 and OutFlow <= 15566.4
458757	0.9680623433187309	Status = '200' and InFlow >= 0.0 and InFlow <= 60968.7
456228	0.9627256843331392	InFlow >= 0.0 and InFlow <= 60968.7 and Status = '200' and OutFlow >= 0.0 and OutFlow <= 15566.4
417662	0.8813442725346703	InFlow >= 0.0 and InFlow <= 60968.7 and UserAgent = 'sls-cpp-sdk v0.6' and Status = '200'
417662	0.8813442725346703	UserAgent = 'sls-cpp-sdk v0.6' and InFlow >= 0.0 and InFlow <= 60968.7
415133	0.8760076135490787	OutFlow >= 0.0 and OutFlow <= 15566.4 and InFlow >= 0.0 and InFlow <= 60968.7 and UserAgent = 'sls-cpp-sdk v0.6' and Status = '200'
415133	0.8760076135490787	OutFlow >= 0.0 and OutFlow <= 15566.4 and UserAgent = 'sls-cpp-sdk v0.6' and InFlow >= 0.0 and InFlow <= 60968.7
415133	0.8760076135490787	OutFlow >= 0.0 and OutFlow <= 15566.4 and UserAgent = 'sls-cpp-sdk v0.6' and Status = '200'
415133	0.8760076135490787	UserAgent = 'sls-cpp-sdk v0.6' and OutFlow >= 0.0 and OutFlow <= 15566.4
414167	0.8739691744110473	InFlow >= 0.0 and InFlow <= 60968.7 and Method = 'PullData' and Status = '200'
414167	0.8739691744110473	Method = 'PullData' and InFlow >= 0.0 and InFlow <= 60968.7

The following table describes the display items.

Display item	Description
count	Number of samples for the current pattern
supportScore	Support level for the current pattern

Display item	Description
pattern	Pattern content, which is organized in the format of conditional queries

9.10. Differential pattern statistical function

Based on the given multi-attribute field samples and conditions, the differential pattern statistical function analyzes the set of differential patterns affecting the conditions. This helps you quickly diagnose the causes for the differences between the conditions.

pattern_diff

Function format:

```
select pattern_diff(array_char_value, array_char_name, array_numeric_value, array_numeric_name, condition, supportScore,posSampleRatio,negSampleRatio)
```

The following table describes the parameters.

Parameter	Description	Value
array_char_value	Input column composed of character type values	Values in array format, for example, array[clientIP, sourceIP, path, logstore]
array_char_name	Name corresponding to the input column composed of character type values	Values in array format, for example, array['clientIP', 'sourceIP', 'path', 'logstore']
array_numeric_value	Input column composed of numeric values	Values in array format, for example, array[Inflow, OutFlow]
array_numeric_name	Name corresponding to the input column composed of numeric values	Values in array format, for example, array['Inflow', 'OutFlow']
condition	Data filtering condition. True indicates positive samples, and False indicates negative samples.	For example: latency ≤ 300
supportScore	Support degree of positive and negative samples for pattern mining	Double type values. Range: (0,1].
posSampleRatio	Sampling ratio of positive samples with a default value of 0.5, which indicates that only half of the positive samples are used	Double type values. Range: (0,1].
negSampleRatio	Sampling ratio of negative samples with a default value of 0.5, which indicates that only half of the negative samples are used	Double type values. Range: (0,1].

Example:

- Statement for query and analysis:

```
* | select pattern_diff(array[ Category, ClientIP, ProjectName, LogStore, Method, Source, UserAgent ], array[ 'Category', 'ClientIP', 'ProjectName', 'LogStore', 'Method', 'Source', 'UserAgent' ], array[ InFlow, OutFlow ], array[ 'InFlow', 'OutFlow' ], Latency > 300, 0.2, 0.1, 1.0) limit 1000
```

- Result:

possupport	posconfidence	negsupport	diffpattern
0.11304206594120514	1.0	0.0	Category = 'sls_operation_log' and ProjectName = 'ali-cn-hangzhou-stg-sls-admin' and LogStore = 'sls_operation_log' and UserAgent = 'ali-log-logtail' and OutFlow >= 4.9E-324 and OutFlow <= 0.0 and InFlow >= 8800.0 and InFlow <= 8850.0
0.11304206594120514	1.0	0.0	ProjectName = 'ali-cn-hangzhou-stg-sls-admin' and LogStore = 'sls_operation_log' and Method = 'PostLogStoreLogs' and Source = '10.206.8.163' and OutFlow >= 4.9E-324 and OutFlow <= 0.0 and InFlow >= 8800.0 and InFlow <= 8850.0
0.11304206594120514	1.0	0.0	Category = 'sls_operation_log' and ProjectName = 'ali-cn-hangzhou-stg-sls-admin' and Method = 'PostLogStoreLogs' and UserAgent = 'ali-log-logtail' and OutFlow >= 4.9E-324 and OutFlow <= 0.0 and InFlow >= 8800.0 and InFlow <= 8850.0
0.11304206594120514	1.0	0.0	Category = 'sls_operation_log' and ProjectName = 'ali-cn-hangzhou-stg-sls-admin' and Method = 'PostLogStoreLogs' and Source = '10.206.8.163' and OutFlow >= 4.9E-324 and OutFlow <= 0.0 and InFlow >= 8800.0 and InFlow <= 8850.0
0.11304206594120514	1.0	0.0	ProjectName = 'ali-cn-hangzhou-stg-sls-admin' and LogStore = 'sls_operation_log' and Source = '10.206.8.163' and UserAgent = 'ali-log-logtail' and OutFlow >= 4.9E-324 and OutFlow <= 0.0 and InFlow >= 8800.0 and InFlow <= 8850.0
0.11304206594120514	1.0	0.0	Category = 'sls_operation_log' and ProjectName = 'ali-cn-hangzhou-stg-sls-admin' and LogStore = 'sls_operation_log' and Source = '10.206.8.163' and OutFlow >= 4.9E-324 and OutFlow <= 0.0 and InFlow >= 8800.0 and InFlow <= 8850.0
0.11304206594120514	1.0	0.0	Category = 'sls_operation_log' and ProjectName = 'ali-cn-hangzhou-stg-sls-admin' and Source = '10.206.8.163' and UserAgent = 'ali-log-logtail' and OutFlow >= 4.9E-324 and OutFlow <= 0.0 and InFlow >= 8800.0 and InFlow <= 8850.0

The following table describes the display items.

Display item	Description
possupport	Support level of positive samples for the mined pattern
posconfidence	Confidence of positive samples for the mined pattern
negsupport	Support level of negative samples for the mined pattern
diffpattern	Content of the mined pattern

9.11. Request URL classification function

The request URL classification function classifies a request URL and attaches a tag to the URL. The function also provides the regular expression that defines the pattern of the tag. You can search for URLs by using the tag pattern and then pass the search result to extract, transform, and load (ETL) processes.

> **Note** The request URL classification function is available in the China (Beijing) and China (Shanghai) regions.

- Function syntax

```
select url_classify(url_path varchar);
select url_classify(url_path varchar, weight long);
```

- Input parameters

Parameter	Description
url_path	The URL of the request.
weight	The number of times that the request URL is called.

- Output parameters

Parameter	Description
url_path	The URL of the request.
api_path	The API endpoint that corresponds to the request URL. The API endpoint is returned by a predefined function.
regex_tpl	The regular expression that is returned by a predefined algorithm.

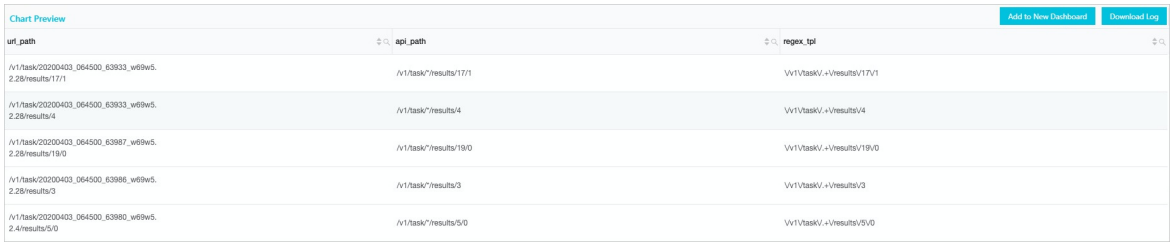
- Response

```
     url_path                           |     api_path                              |          regex_
tpl
-----------------------------------+-----------------------------+-----------------------------
--------
 /gl/balance/666398186799140           | /gl/balance/*                              | \/gl\/balance
\/[0-9].+
 /gl/glaccount/30579281472076          | /gl/glaccount/*                           | \/gl\/glaccount
\/[0-9].+
 /gl/balance/709016207098025           | /gl/balance/*                              | \/gl\/balance\
/[0-9]. +
```

- Example
  - Enter the following query statement:

```
* | select url_classify(uri, num) from (select uri, COUNT(*) as num from log group by uri limit
1000)
```

  - View the query result.



9.12. Root cause analysis function

Log Service provides powerful alerting and analysis capabilities that help you quickly analyze and locate the subdimensions of abnormal metrics. When a time series metric is abnormal, you can use the root cause analysis function to quickly analyze the dimension attributes that result in the abnormal metric.

rca_kpi_search

Function format:

```
select rca_kpi_search(varchar_array, name_array, real, forecast, level)
```

The following table lists the parameters of the function.

Parameter	Description	Value
varchar_array	The dimensions.	Array. Example: array[col1, col2, col3].

Parameter	Description	Value
name_array	The dimension attributes.	Array. Example: array['col1', 'col2', 'col3'].
real	The actual value of each dimension specified by varchar_array.	Double type. Valid values: all real numbers.
forecast	The predicted value of each dimension specified by varchar_array.	Double type. Valid values: all real numbers.
level	The number of dimensions corresponding to the output root cause sets. A value of 0 indicates that all root cause sets that are found are returned.	Long type. Valid values: [0, number of analyzed dimensions]. The number of analyzed dimensions is the number of elements in the array specified by the varchar_array parameter.

Example:

- The query statement is as follows.

  The query statement uses a subquery to obtain the actual value and predicted value of each fine-grained attribute, and then calls the rca_kpi_search function to analyze the root cause of the exception.

```
* not Status:200 |
select rca_kpi_search(
 array[ ProjectName, LogStore, UserAgent, Method ],
 array[ 'ProjectName', 'LogStore', 'UserAgent', 'Method' ], real, forecast, 1)
from (
select ProjectName, LogStore, UserAgent, Method,
 sum(case when time < 1552436040 then real else 0 end) * 1.0 / sum(case when time < 1552436040
then 1 else 0 end) as forecast,
 sum(case when time >=1552436040 then real else 0 end) *1.0 / sum(case when time >= 1552436040
then 1 else 0 end) as real
 from (
select __time__ - __time__ % 60 as time, ProjectName, LogStore, UserAgent, Method, COUNT(*) as rea
l
from log GROUP by time, ProjectName, LogStore, UserAgent, Method )
GROUP BY ProjectName, LogStore, UserAgent, Method limit 100000000)
```

- The following figure shows the output result.

The following figure shows the structure of the output result.

```
{
  "rcSets": [
    {
      "rcItems": [
        {
          "kpi": [{"attr": "xxx", "val": "xxx"}],
          "nleaf": 100,
          "change": 0.524543,
          "score": 0.1454543
        }
      ]
    }
  ]
}
```

The following table describes the display items.

Display item	Description
rcSets	The root cause sets. Each value is an array.
rcItems	The root cause set.
kpi	The KPI in the root cause set, which is an array. Each value in the array is in JSON format. attr indicates a dimension, and val indicates an attribute in the dimension.
nleaf	The number of leaves that the current KPI covers in the raw data. **Note** A leaf is a log for the finest-grained attributes.
change	The ratio of changes in the leaves covered by the current KPI to the total changes at the same time point.
score	The abnormality score of the current KPI. Valid values: [0, 1].

The output result is in JSON format as follows:

```
{
  "rcSets": [
  {
    "rcItems": [
    {
      "kpi": [
      {
        "attr": "country",
        "val": "*"
      },
      {
        "attr": "province",
        "val": "*"
      },
      {
        "attr": "provider",
        "val": "*"
      },
      {
        "attr": "domain",
        "val": "example.com"
      },
      {
        "attr": "method",
        "val": "*"
      }
      ],
      "nleaf": 119,
      "change": 0.3180687806279939,
      "score": 0.14436007709620113
    }
    ]
  }
  ]
}
```


9.13. Correlation analysis functions

You can use a correlation analysis function to quickly find the metrics that are correlated with a specified metric or time series data among multiple observed metrics in the system.

Function list

Function	Description
ts_association_analysis	Quickly finds the metrics that are correlated with a specified metric among multiple observed metrics in the system.
ts_similar	Quickly finds the metrics that are correlated with specified time series data among multiple observed metrics in the system.

ts_association_analysis

Function format:

```
select ts_association_analysis(stamp, params, names, indexName, threshold)
```

The following table describes the parameters.

Parameter	Description	Value
stamp	The Unix timestamp.	Long type.
params	The dimensions of the metrics to be analyzed.	Array of the double type. For example, Latency, QPS, and NetFlow.
names	The names of the metrics to be analyzed.	Array of the varchar type. For example, Latency, QPS, and NetFlow.
indexName	The name of the target metric.	Varchar type, for example, Latency.
threshold	The threshold of correlation between the metrics to be analyzed and the target metric.	Double type. Valid values: [0, 1].

Result:

- name: the name of the analyzed metric.
- score: the value of correlation between the analyzed metric and the target metric. Valid values: [0, 1].

Sample code:

```
* | select ts_association_analysis(
              time,
              array[inflow, outflow, latency, status],
              array['inflow', 'outflow', 'latency', 'status'],
              'latency',
              0.1) from log;
```

Sample result:

```
| results               |
| --------------------- |
| ['latency', '1.0']    |
| ['outflow', '0.6265'] |
| ['status', '0.2270']  |
```

ts_similar

Function format 1:

```
select ts_similar(stamp, value, ts, ds)
select ts_similar(stamp, value, ts, ds, metricType)
```

The following table describes the parameters.

Parameter	Description	Value
stamp	The Unix timestamp.	Long type.
value	The value of the specified metric.	Double type.

Parameter	Description	Value
ts	The sequence of time for the specified curve.	Array of the double type.
ds	The sequence of numeric data for the specified curve.	Array of the double type.
metricType	The type of correlation between the measured curves.	Varchar type. Valid values: SHAPE, RMSE, PEARSON, SPEARMAN, R2, and KENDALL

Function format 2:

```
select ts_similar(stamp, value, startStamp, endStamp, step, ds)
select ts_similar(stamp, value, startStamp, endStamp, step, ds, metricType )
```

The following table describes the parameters.

Parameter	Description	Value
stamp	The Unix timestamp.	Long type.
value	The value of the specified metric.	Double type.
startStamp	The start timestamp of the specified curve.	Long type.
endStamp	The end timestamp of the specified curve.	Long type.
step	The time interval between two adjacent points in the sequence of time.	Long type.
ds	The sequence of numeric data for the specified curve.	Array of the double type.
metricType	The type of correlation between the measured curves.	Varchar type. Valid values: SHAPE, RMSE, PEARSON, SPEARMAN, R2, and KENDALL

Result:

score: the value of correlation between the analyzed metric and the target metric. Valid values: [-1, 1].

Sample code:

```
* | select vhost, metric, ts_similar(time, value, 1560911040, 1560911065, 5, array[5.1,4.0,3.3,5.6,4.0,7.2], 'PEARSON') from log  group by vhost, metric;
```

Sample result:

```
| vhost  | metric          | score               |
| ------ | --------------- | ------------------- |
| vhost1 | redolog         | -0.3519082537204182  |
| vhost1 | kv_qps          | -0.15922168009772697 |
| vhost1 | file_meta_write | NaN                 |
```

9.14. Kernal density estimation functions

Kernel density estimation is a non-parametric test method. It is used to estimate unkonwn density functions in probability theory.

Kernel density estimation functions use a smooth peak function to simulate the real probability distribution curve by fitting the observed data points.

- Function format:

```
select kernel_density_estimation(bigint stamp, double value, varchar kernelType)
```

- Parameters

Parameter	Description
stamp	Unix timestamp. Unit: second.
value	Observed value.
kernelType	○ box: rectangle. ○ epanechnikov: Epanechnikov curve. ○ gaussian: Gaussian curve.

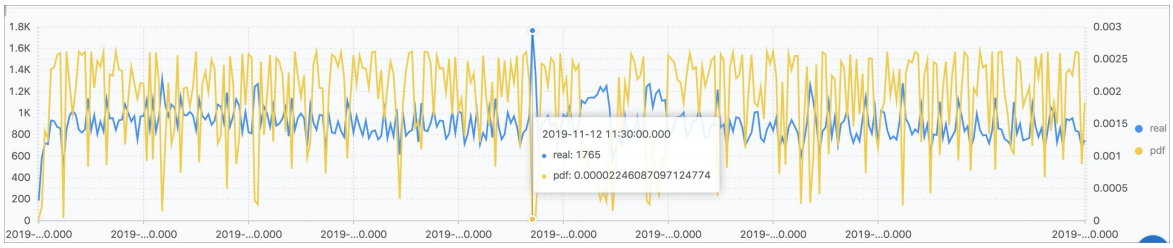
- Output result

Display item	Description
unixtime	The time of the source data.
real	Observed value.
pdf	The probability of each point.

- Examples
  - Sample code:

```
* |
select
    date_trunc('second', cast(t1[1] as bigint)) as time, t1[2] as real, t1[3] as pdf from (
        select kernel_density_estimation(time, num, 'gaussian') as res from (
            select __time__ - __time__ % 10 as time, COUNT(*) * 1.0 as num from log group by tim
e order by time)
        ), unnest(res) as t(t1)  limit 1000
```

- Sample result:



9.15. Time series padding function

The time series padding function pads data points that are missing in a time series.

- Function expressions

```
select series_padding(long stamp, double value, long interval, varchar padType)
```

- Input parameters

Parameter	Description
stamp	The UNIX timestamp of the data.
value	The numeric data that corresponds to each specified point in time.
interval	The interval at which data is collected. For example, if data is collected every 10 seconds, the interval is 10.
padType	The type of padded data points. Valid values: zero, mean, forward, and backward. ○ zero: pads 0. ○ mean: pads the average of the valid values on both sides of a missing data point. ○ forward: pads the valid value on the left of a missing point. ○ backward: pads the valid value on the right of a missing point.

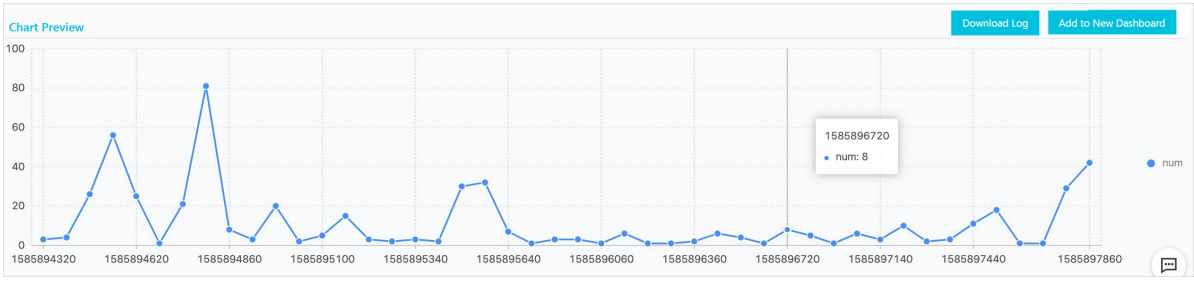
- Result

```
unixtime   |       pad_value
-----------+-----------------------
 1.5513696E9 |   0.11243584740434608
 1.5513732E9 |   0.09883780706698506
 1.5513768E9 |   0.08240823914341992
 1.5513804E9 |    0.0728240514818139
  1.551384E9 |   0.05888517541914705
 1.5513876E9 |   0.04953931499029833
 1.5513912E9 |  0.043698605551761895
 1.5513948E9 |   0.04400292632222124
 1.5513984E9 |   0.04727081764249449
  1.551402E9 |  0.054632234293121314
 1.5514056E9 |   0.05331214064978596
 1.5514092E9 |   0.05093117289934144
 1.5514128E9 |  0.053620170319174806
 1.5514164E9 |   0.05405914786225842
```

- Examples

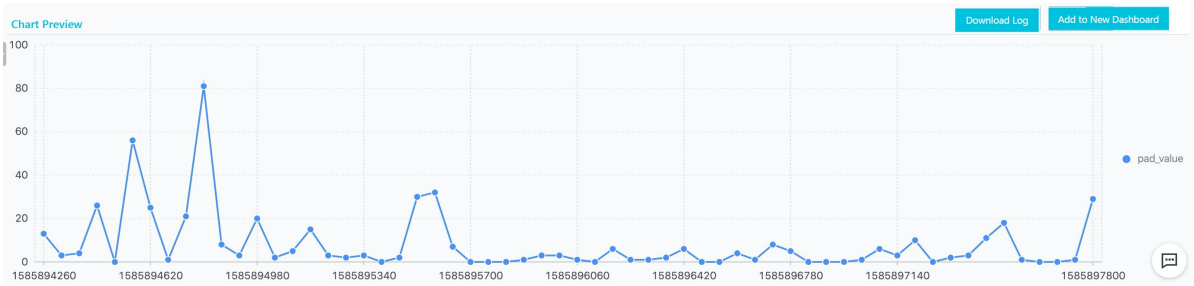
  Execute the following query statement and select the line chart to display the query results. Several data points are missing on the line chart, as shown in the following figure.

```
* and Method: GetLogStoreLogs and ProjectName: lunar and LogStore: geos and Latency > 800000 | sel
ect __time__ - __time__ 60% as time, COUNT(*) * 1.0 as num from log group by time order by time as
c limit 1000
```



Execute the following query statement that includes the time series padding function, and select the line chart to display the query results. The missing data points are padded, as shown in the following figure.

```
* and Method: GetLogStoreLogs and ProjectName: lunar and LogStore: geos and Latency > 800000 | sel
ect series_padding(time, num, 60, 'zero') from (select __time__ - __time__ 60% as time, COUNT(*) *
1.0 as num from log group by time order by time asc limit 1000)
```



9.16. Anomaly comparison function

The anomaly comparison function compares the degree of differences of an observation object in two time ranges.

- Function syntax 1
  - Function expressions

    ```
    select anomaly_compare(long stamp, array[ feature_1, feature_2 ], long timePoint, long interval)
    select anomaly_compare(long stamp, array[ feature_1, feature_2 ], array[ feature1_name, feature2
    _name ], long timePoint, long interval)
    ```

  - Input parameters

Parameter	Description
stamp	The UNIX timestamp of the data.
array[features]	The metrics of the observation object at a specific point in time.
array[featureNames]	The description of the metrics.
timePoint	The UNIX timestamp of the time when the observed object changes.
interval	The interval at which data is collected. For example, if data is collected every 10 seconds, the interval is 10.

- Function syntax 2

- Function expressions

```
select anomaly_compare(long stamp, array[ feature_1, feature_2 ], array[ feature1_name, feature2_name ], long version)
```

- Input parameters

Parameter	Description
stamp	The UNIX timestamp of the data.
array[features]	The metrics of the observation object at a specific point in time.
array[featureNames]	The description of the metrics.
version	The version number of the time series. ■ A value of 0 indicates the raw data. ■ A value of 1 indicates the new data.

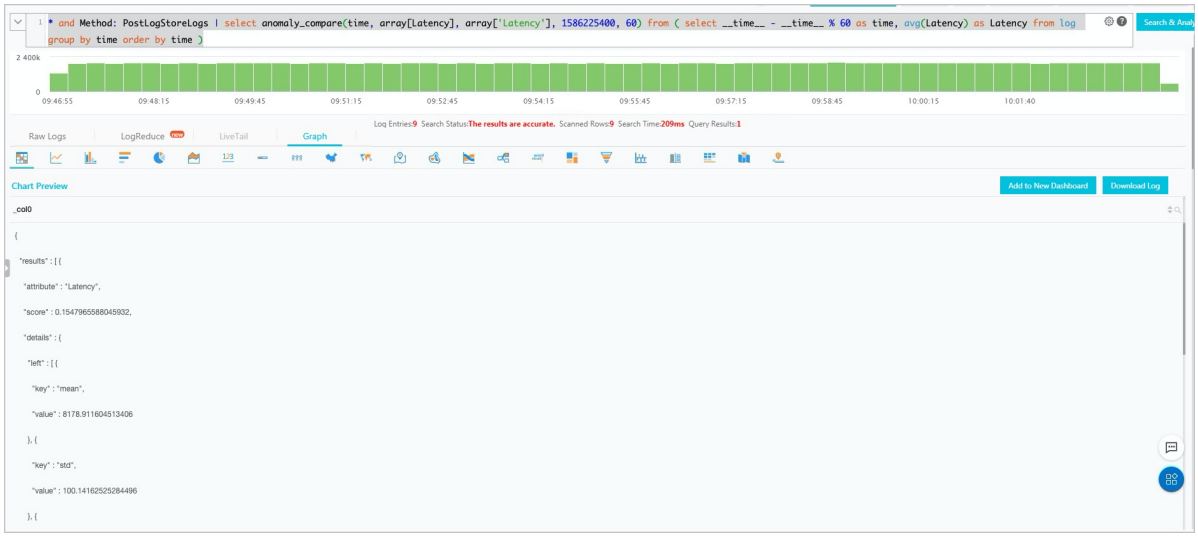
- Result

```
{
   "results" : [ {
     "attr" : "cpu",
     "anomalyScore" : 0.01106371634297909,
     "details" : {
       "left" : [ {
         "key" : "mean",
         "value" : 0.07002069952622482
       }, {
         "key" : "std",
         "value" : 0.1364542814430179
       }, {
         "key" : "median",
         "value" : 0.04467685956328345
       }, {
         "key" : "variance",
         "value" : 0.018619770924130346
       } ],
       "rightMetrics" : [ {
         "key" : "mean",
         "value" : 0.4472823405432968
       }, {
         "key" : "std",
         "value" : 0.22405908739288383
       }, {
         "key" : "median",
         "value" : 0.42513225830553775
       }, {
         "key" : "variance",
         "value" : 0.05020247464333195
       } ]
     }
   } ]
 }
```

- Result description
  - The mean, std, median, and variance methods are used for the statistics of a time series.

- If you specify the names of metrics, the names are included in the attr field. Otherwise, the prefix column_ is concatenated with the array subscript of a metric as the name of the metric, for example, column_0.
- The anomalyScore indicates the degree of difference of a feature metric. Value values: 0 to 1. If the value approaches 0, the degree of difference is low. If the value approaches 1, the degree of difference is high.

- Example



10.Analyze logs by using the JDBC API

You can use the Java Database Connectivity (JDBC) API to connect a database such as a MySQL database to Log Service. Then, you can use the SQL-92 syntax to query and analyze log data.

Prerequisites

- An AccessKey pair is created for an Alibaba Cloud account or a RAM user. For more information, see AccessKey pair.

  If you use the AccessKey pair of a RAM user, the RAM user must belong to the Alibaba Cloud account to which the project that you want to connect belongs. The RAM user must be granted the read permissions on the project.
- A Logstore is created. For more information, see Create a Logstore.

Supported version

Log Service supports only JDBC 5.1.49.

Parameters

This section describes the parameters that you must set if you want to connect a MySQL database to Log Service.

> **Note** If you connect a MySQL database to Log Service by using the JDBC API and then query data, the query result cannot be paginated.

- Syntax

```
mysql -hhost -user -password -port
use database;
```

- Example

```
mysql -hmy-project.cn-hangzhou-intranet.log.aliyuncs.com -ubq****mo86kq -p4f****uZP -P10005
use my-project;
```

- Parameters

Parameter	Description
host	The endpoint of Log Service. You must add the project name to the endpoint, for example, my-project.cn-hangzhou-intranet.log.aliyuncs.com. You can connect to Log Service only over the classic network or a virtual private cloud (VPC). For more information, see Internal Log Service endpoints.
port	The port that you can use to connect to Log Service. Default value: 10005.
user	The AccessKey ID of your Alibaba Cloud account.
password	The AccessKey secret of your Alibaba Cloud account.
database	The name of the Log Service project.
table	The name of the Log Service Logstore. **Note** You must specify the Logstore in a query statement.

Search and analysis syntax

- Filtering syntax

  > **Note** You must include the __date__ or __time__ field in a WHERE clause to limit the time range of a query. The data type of the __date__ field is timestamp, and the data type of the __time__ field is bigint. Examples:
  > - __date__ > '2017-08-07 00:00:00' and __date__ < '2017-08-08 00:00:00'
  > - __time__ > 1502691923 and __time__ < 1502692923

  The following table describes the filtering syntax of a WHERE clause.

Semantics	Example	Description
String search	**key = "value"**	Queries data after word-delimiting.
String fuzzy search	○ **key has 'valu*'** ○ **key like 'value_%'**	Queries data in fuzzy match mode after word-delimiting.
Value comparison	**num_field > 1**	The comparison operators include greater than (>), greater than or equal to (>=), equal to (=), less than (<), and less than or equal to (<=).
Logical operation	**and or not**	Examples: **a = "x" and b ="y"** or **a = "x" and not b ="y"**.
Full-text search	**__line__ ="abc"**	If you perform a full-text search, you must use the __line__ key.

- Calculation syntax

  Calculation operators are supported. For more information, see Log analysis overview.

- SQL-92 syntax

  The SQL-92 syntax includes the filtering syntax and calculation syntax. Example:

```
status>200 |select avg(latency),max(latency) ,count(1) as c GROUP BY  method  ORDER BY c DESC  LIMIT 20
```

  You can combine the analytic statement in the preceding query statement with a time condition expression and include the statement and expression in a WHERE clause as a search condition. This clause complies with the SQL-92 syntax, as shown in the following statement:

```
select avg(latency),max(latency) ,count(1) as c from sample-logstore where status>200 and __time__ >=1500975424 and __time__ < 1501035044 GROUP BY  method  ORDER BY c DESC  LIMIT 20
```

Access Log Service by using the JDBC API

- Use an application to query data in Log Service

  You can use an application that supports the MySQL connector to connect to Log Service. Then, you can use the MySQL syntax to query data in Log Service. You can use the JDBC API or MySQLdb API for Python to connect to Log Service. The following script shows how to use the JDBC API to connect to Log Service:

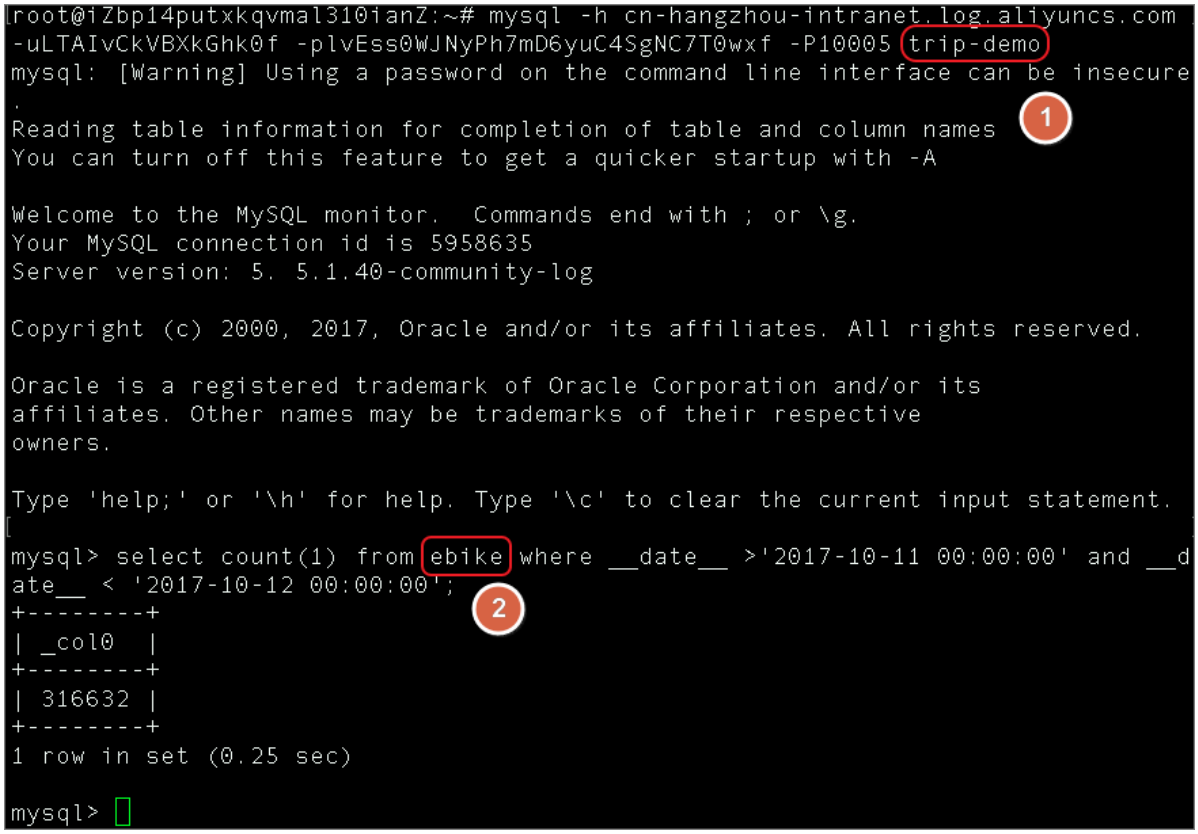
```
import com.mysql.jdbc.*;
import java.sql.*;
import java.sql.Connection;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
```

```
import java.sql.Statement;
public class testjdbc {
    public static void main(String args[]){
        Connection conn = null;
        Statement stmt = null;
        try {
            //STEP 2: Register JDBC driver
            Class.forName("com.mysql.jdbc.Driver");
            //STEP 3: Open a connection
            System.out.println("Connecting to a selected database...");
            conn = DriverManager.getConnection("jdbc:mysql://projectname.cn-hangzhou-intranet.log.
aliyuncs.com:10005/sample-project","accessid","accesskey");
            System.out.println("Connected database successfully...");
            //STEP 4: Execute a query
            System.out.println("Creating statement...");
            stmt = conn.createStatement();
            String sql = "SELECT method,min(latency,10)  as c,max(latency,10) from sample-logstore
where  __time__>=1500975424 and __time__ < 1501035044 and latency > 0  and latency < 6142629 and
not (method='Postlogstorelogs' or method='GetLogtailConfig') group by method " ;
            String sql_example2 = "select count(1) ,max(latency),avg(latency), histogram(method),h
istogram(source),histogram(status),histogram(clientip),histogram(__source__) from  test10 where __
date__  >'2017-07-20 00:00:00'  and  __date__ <'2017-08-02 00:00:00' and __line__='abc#def' and la
tency < 100000 and (method = 'getlogstorelogS' or method='Get**' and method <> 'GetCursorOrData' )
";
            String sql_example3 = "select count(1) from  sample-logstore where     __date__  >
'2017-08-07 00:00:00' and  __date__ <     '2017-08-08 00:00:00' limit 100";
            ResultSet rs = stmt.executeQuery(sql);
            //STEP 5: Extract data from result set
            while(rs.next()){
                //Retrieve by column name
                ResultSetMetaData data = rs.getMetaData();
                System.out.println(data.getColumnCount());
                for(int i = 0;i < data.getColumnCount();++i) {
                    String name = data.getColumnName(i+1);
                    System.out.print(name+":");
                    System.out.print(rs.getObject(name));
                }
                System.out.println();
            }
            rs.close();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (stmt != null) {
                try {
                    stmt.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
```

```
            }
        }
    }
}
```

- Use a tool to connect to Log Service

  Use a MySQL client to connect to Log Service over the classic network or a VPC.



  - Enter your project name at 1.
  - Enter your Logstore name at 2.

11.Advanced analysis

11.1. Case study

This topic describes cases of log data analysis.

Case list

- Trigger an alert when the error rate exceeds 40% over the last five minutes
- Count traffic and configure alerts
- Calculate the average latency of each data interval
- Return the percentages of different results
- Count the number of logs that meet the query condition

Trigger an alert when the error rate exceeds 40% over the last five minutes

Calculate the percentage of error 500 every minute. An alert is triggered when the error rate exceeds 40% over the last five minutes.

```
status:500 | select __topic__, max_by(error_count,window_time)/1.0/sum(error_count) as error_ratio, s
um(error_count) as total_error  from (
select __topic__, count(*) as error_count , __time__ - __time__ % 300  as window_time  from log group
by __topic__, window_time
)
group by __topic__  having  max_by(error_count,window_time)/1.0/sum(error_count)   > 0.4  and sum(err
or_count) > 500 order by total_error desc limit 100
```

Count traffic and configure alerts

Count traffic every minute. An alert is triggered when traffic plunges. Traffic counted in the last minute does not cover a full minute. Therefore, divide the statistical value by using `greatest(max(__time__) - min(__time__),1)` for normalization to count the average traffic per minute.

```
* | SELECT SUM(inflow) / greatest(max(__time__) - min(__time__),1)  as inflow_per_minute, date_trunc(
'minute',__time__)  as minute group by minute
```

Calculate the average latency of each data interval

Calculate the average latency of each bucket set by data interval.

```
* | select avg(latency) as latency , case when originSize < 5000 then 's1' when originSize < 20000 th
en 's2' when originSize < 500000 then 's3' when originSize < 100000000 then 's4' else 's5' end as os
group by os
```

Return the percentages of different results

Return the count results of different departments and the percentages of these results. This query combines the subquery and window functions. `sum(c) over()` indicates the sum of values in all rows.

```
* |  select  department, c*1.0/ sum(c) over ()  from(select  count(1) as c, department  from log gr
oup by department)
```

Count the number of logs that meet the query condition

URLs must be by counted by characteristics. In this situation, you can use the CASE WHEN syntax and the simpler count_if syntax.

```
* | select  count_if(uri like '%login') as login_num, count_if(uri like '%register') as register_num, date_format(date_trunc('minute', __time__), '%m-%d %H:%i') as time  group by time order by time limit 100
```

11.2. Optimize queries

This topic describes how to optimize queries to improve query efficiency.

Increase the number of shards

Shards are computing resources. The calculation speed increases with the number of shards. You must ensure that the system scans only up to 50 million data entries per shard. You can split a shard to increase the number of shards. For more information, see Split a shard.

> **Notice** You are additionally charged for the shards that are generated by splitting a shard. After you split a shard, your queries are accelerated only when you query new data. Existing data remains in the original shard.

Shorten the time range and reduce the data volume of a query

- A long time range slows down a query.

  If you want to accelerate calculation, you can shorten the time range of a query.

- A large data volume slows down a query.

  We recommend that you reduce the data volume of a query.

Query data repeatedly

If the result of a query is inaccurate, you can query the data repeatedly. Each time you perform a query, the underlying acceleration mechanism analyzes the data based on the existing query results. Therefore, repeated queries can return more accurate results.

Optimize analytic statements

A time-consuming query statement features the following characteristics:

- The GROUP BY clause is used to group analysis results based on one or more columns of the string type.
- The GROUP BY clause is used to group analysis results based on more than five columns.
- Operations that generate strings are included in the analytic statement.

You can use the following methods to optimize analytic statements:

- Do not include operations that generate strings.

  For example, if you use the date_format function to generate a timestamp in a specified format, the query is inefficient. We recommend that you use the date_trunc function or the time_series function to generate a timestamp.

  ```
  * | select date_format(from_unixtime(__time__) , '%H_%i') as t, count(1) group by t
  ```

- Do not group analysis results based on one or more columns of the string type.

  For example, if you use the GROUP BY clause to group analysis results based on one or more columns of the string type, the workload for hash calculation is heavy. The workload for hash calculation accounts for more than 50% of the overall workload for calculation. Examples:

- Efficient query statement

```
* | select count(1) as pv , from_unixtime(__time__-__time__%3600) as time group by __time__-__ti
me__%3600
```

- Inefficient query statement

```
* | select count(1) as pv , date_trunc('hour',__time__) as time group by time
```

The two query statements are used to calculate the number of logs per hour. In the second statement, the timestamps are converted to strings. Then, the analysis results are grouped based on the strings. For example, 2021-12-12 00:00:00 is a timestamp in the string format. In the first statement, the timestamps on the hour are obtained, the analysis results are grouped based on the timestamps, and then the timestamps are converted to strings.

- If you want to group analysis results based on multiple columns, we recommend that you place a field that has a larger number of values before a field that has a smaller number of values.

  For example, if the number of values for a specified field is 13 and the number of values for the uid field is 100 million, we recommend that you place the uid field before the specified field in the GROUP BY clause. Examples:

  - Efficient query statement

```
* | select province,uid,count(1) group by uid,province
```

  - Inefficient query statement

```
* | select province,uid,count(1) group by province,uid
```

- Use approximate functions.

  The performance of approximate functions is better than the performance of functions with specified precision. Approximate functions sacrifice precision for speed. Examples:

  - Efficient query statement

```
* |select approx_distinct(ip)
```

  - Inefficient query statement

```
* | select count(distinct(ip))
```

- Specify only the columns that you want to query in an analytic statement.

  In an analytic statement, we recommend that you query only the columns that are required in calculation. If you want to query all columns, use the search syntax. Examples:

  - Efficient query statement

```
* |select a,b c
```

  - Inefficient query statement

```
* |select *
```

- Place the columns that are not used for grouping in an aggregate function.

  For example, the values of the userid and username columns must match each other. You can use the GROUP BY clause to group only the values of the userid column. Examples:

  - Efficient query statement

```
* | select userid, arbitrary(username), count(1) group by userid
```

- Inefficient query statement

```
* | select userid, username, count(1) group by userid,username
```

- Do not use the IN clause.

  We recommend that you do not use the IN clause in an analytic statement. You can use OR in a search statement. Examples:

  - Efficient query statement

```
key: a or key: b or key: c | select count(1)
```

  - Inefficient query statement

```
* | select count(1) where key in ('a','b')
```

11.3. Time field conversion examples

During query and analysis, you often need to process time fields in logs, such as converting a timestamp to a specified format. This topic uses some examples to describe how to convert time fields.

A log may include multiple fields that record the time. For example:

- `__time__`: the log time that you specify when you use the API or SDK to write log data. This field can be used for log data shipping, query, and analysis.
- Original time field in the log: the field that records the log event occurrence time when the log is generated. This field is in the raw log.

Time fields in different formats are difficult to view and read. In this case, you can convert the time fields to the specified format in query and analysis. For example:

1. Convert __time__ to a timestamp
2. Display __time__ in a specified format
3. Convert a timestamp to a specified format

Convert __time__ to a timestamp

We recommend that you use the from_unixtime function to convert the `__time__` field to a timestamp.

```
* | select from_unixtime(__time__)
```

Display __time__ in a specified format

Display the `__time__` field in the format of `YYYY-MM-DD HH:MM:SS`. We recommend that you use the date_format function to convert the field.

```
* | select date_format(__time__, '%Y-%m-%d %H:%i:%S')
```

Convert the time in a log to a specified format

Convert the time field in a log to the specified format (`YYYY-MM-DD HH:MM:SS`), and perform the GROUP BY operation on the `YYYY-MM-DD` part. We recommend that you use the date_format function to convert the field.

- Sample log:

```
__topic__:
body_byte_sent:  307
hostname:  example.com
http_user_agent:  Mozilla/5.0 (iPhone; CPU iPhone OS 10_3_3 like Mac OS X) AppleWebKit/603.3.8 (KH
TML, like Gecko) Mobile/14G60 QQ/192.0.2.1 V1_IPH_SQ_7.1.8_1_APP_A Pixel/750 Core/UIWebView NetTyp
e/WIFI QBWebViewType/1
method:  GET
referer:  www.example.com
remote_addr:  192.0.2.0
request_length:  111
request_time:  2.705
status:  200
upstream_response_time:  0.225582883754
url:  /?k0=v9&
time:2017-05-17 09:45:00
```

- SQL statement example:

```
* | select date_format (date_parse(time,'%Y-%m-%d %H:%i:%S'), '%Y-%m-%d') as day, count(1) as uv g
roup by day order by day asc
```

12.Associate Log Service with external data sources

12.1. Overview

Log Service provides the external storage feature. You can use the feature to associate Log Service with MySQL databases, Alibaba Cloud Object Storage Service (OSS) buckets, or hosted CSV files. This topic describes the scenarios, benefits, and supported external stores of the external storage feature.

Scenarios

When you analyze data, you may need to obtain different types of data from separate storage resources. For example, you need to obtain data of user operations and user behavior from Log Service, and obtain data of user properties, registration, funds, and props from a database. In this example, you need to classify and analyze data and then write the analysis results to the report system of the database.

To do this, you can migrate data to a centralized storage system and then analyze the data. However, the migration process is time-consuming and labor-intensive. Data must be cleansed and formatted during migration, and network resources are consumed. To address these issues, Log Service provides API operations for external storage. You can call the API operations to achieve the following goals:

- Define mappings between data in external stores and data in Log Service. Data migration is not required.
- Use a unified query engine. You can use JOIN statements to perform JOIN queries on data in Log Service and data in external stores.
- Store query results in external stores.

Benefits

- Cost-effective
  - The external storage feature eliminates the need for data migration, which further reduces your overall costs. Data in different storage systems are stored in different formats. The API operations that you can call to manage data also vary based on the storage systems. This results in complicated data conversion during data migration. If you use the external storage feature of Log Service, you do not need to migrate data.
  - The external storage feature eliminates the need for data maintenance, which further reduces your overall costs. If you migrate data, you must update and maintain the data at the earliest opportunity.
- Convenient
  - You can use SQL statements to analyze data and obtain the analysis results within seconds.
  - You can add charts to a dashboard and view the charts when you open the dashboard.

Supported external stores

The external storage feature of Log Service allows you to associate Log Service with MySQL databases, OSS buckets, or hosted CSV files. The following table describes the supported external stores.

Supported external store	Read from the external store	Write to the external store	Method to create an external store	Supported region
MySQL databases	Supported	Supported	API, SDK, and CLI	All regions
OSS buckets	Supported	Supported	SQL create table	All regions
Hosted CSV files	Supported	Not supported	SDK	China (Shanghai) and Russia (Moscow)

12.2. Associate Log Service with a MySQL database

This topic describes how to create an external store to associate Log Service with a MySQL database.

Prerequisites

- Data is collected and stored in Log Service. For more information, see Data collection overview.
- Data is stored in the MySQL database.

Context

The external storage feature of Log Service allows you to associate Log Service with databases created on ApsaraDB RDS for MySQL instances, self-managed MySQL databases hosted on Elastic Compute Service (ECS) instances, and self-managed MySQL databases created in other scenarios. The external storage feature also allows you to write query and analysis results to the MySQL databases for further processing. In the following descriptions, the ApsaraDB RDS for MySQL instances are referred to as RDS instances. For more information about how to create a MySQL external store, see Associate a Logstore with a MySQL database to perform query and analysis.

Procedure

1. Configure a whitelist for your MySQL database.
   - If you use a MySQL database created on an RDS instance, add the following CIDR blocks to the whitelist: 100.104.0.0/16, 11.194.0.0/16, and 11.201.0.0/16. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for MySQL instance.
   - If you use a self-managed MySQL database hosted on a VPC-type ECS instance and the ECS instance is added to a security group, configure security group rules to allow access from the following CIDR blocks: 100.104.0.0/16, 11.194.0.0/16, and 11.201.0.0/16. For more information, see Add a security group rule.
   - If you use a self-managed MySQL database created in other scenarios, add the following CIDR blocks to the whitelist: 100.104.0.0/16, 11.194.0.0/16, and 11.201.0.0/16.
2. Create an external store.
   i. Install the Log Service CLI. For more information, see CLI Overview.
   ii. Create a configuration file named */root/config.json*.
   iii. Add the following script to the */root/config.json* file. Replace the parameter values based on your business requirements.

```
{
"externalStoreName":"storename",
"storeType":"rds-vpc",
"parameter":
   {
   "region":"cn-qingdao",
   "vpc-id":"vpc-m5eq4irc1pucp*******",
   "instance-id":"i-m5eeo2whsn*******",
   "host":"localhost",
   "port":"3306",
   "username":"root",
   "password":"****",
   "db":"scmc",
   "table":"join_meta"
   }
}
```

Parameter	Description
externalStoreName	The name of the external store. The name must be in lowercase.
storeType	The type of the data source. Set the value to rds-vpc.
region	The region. ■ If you use a MySQL database created on an RDS instance, set region to the region where the RDS instance resides. ■ If you use a self-managed MySQL database hosted on a VPC-type ECS instance, set region to the region where the ECS instance resides. ■ If you use a self-managed MySQL database created in other scenarios, set region to an empty string. Format: **"region": ""**.
vpc-id	The ID of the VPC. ■ If you use a MySQL database created on a VPC-type RDS instance, set vpc-id to the ID of the VPC to which the RDS instance belongs. ■ If you use a self-managed MySQL database hosted on a VPC-type ECS instance, set vpc-id to the ID of the VPC to which the ECS instance belongs. ■ If you use a MySQL database created on a classic network-type RDS instance or if you use a self-managed MySQL database created in other scenarios, set vpc-id to an empty string. Format: **"vpc-id": ""**.
instance-id	The ID of the instance. ■ If you use a MySQL database created on an RDS instance, set instance-id to the value of the VpcCloudInstanceId parameter that is specified for the RDS instance. You can call the DescribeDBInstanceAttribute operation to obtain the value of the VpcCloudInstanceId parameter. ■ If you use a self-managed MySQL database hosted on a VPC-type ECS instance, set instance-id to the ID of the ECS instance. ■ If you use a self-managed MySQL database created in other scenarios, set instance-id to an empty string. Format: **"instance-id": ""**.
host	The address of your MySQL database. ■ If you use a MySQL database created on a VPC-type RDS instance, set host to an internal endpoint of the RDS instance. ■ If you use a self-managed MySQL database hosted on a VPC-type ECS instance, set host to the private IP address of the ECS instance. ■ If you use a self-managed MySQL database created in other scenarios, set host to a host address of the database. Make sure that the host address is accessible.

Parameter	Description
port	The port number. ■ If you use a MySQL database created on an RDS instance, set port to the port of the RDS instance. ■ If you use a self-managed MySQL database hosted on a VPC-type ECS instance, set port to the MySQL service port of the ECS instance. ■ If you use a self-managed MySQL database created in other scenarios, set port to the MySQL service port.
username	The username of the account that you use to log on to your MySQL database.
password	The password of the account that you use to log on to your MySQL database.
db	The name of your MySQL database.
table	The name of the table that you want to use in your MySQL database.

iv. Create an external store.

Replace the value of project_name with the name of the project that you want to use.

```
aliyunlog log create_external_store --project_name="log-rds-demo" --config="file:///root/config.json"
```

What to do next

- Update the MySQL external store.

```
aliyunlog log update_external_store --project_name="log-rds-demo" --config="file:///root/config.json"
```

- Delete the MySQL external store.

```
aliyunlog log delete_external_store --project_name="log-rds-demo" --store_name=abc
```

What's next

Join queries on a Logstore and a MySQL database

12.3. Associate Log Service with an OSS bucket

This topic describes how to create an external store to associate Log Service with an Object Storage Service (OSS) bucket.

Prerequisites

- Logs are collected. For more information, see Data collection overview.
- The indexing feature is enabled, and indexes are created. For more information, see Configure indexes.
- An OSS bucket is created. For more information, see Create buckets.
- CSV files are uploaded to the OSS bucket. For more information, see Upload objects.

Benefits

If you associate Log Service with OSS buckets to perform query and analysis, you can receive the following benefits:

- Cost-effectiveness: If you store infrequently updated data in OSS buckets, the data can be read over an internal network. In this case, you need only to pay for the storage service, and you are not charged for Internet

traffic.

- Reduced O&M workload: You can perform lightweight association analysis without the need to store all data in one storage system.
- High efficiency: You can use SQL statements to analyze data and view the analysis results within seconds. You can also create charts based on analysis results that are commonly queried. Then, you can click the charts to view the analysis results.

Procedure

1.
2.
3.
4. On the page that appears, enter a query statement in the search box and click **Search & Analyze**.

   Execute the following SQL statement to create a virtual external table named user_meta1 and map the table to the OSS object user.csv. If **result** in the output is **true**, the SQL statement is successfully executed, and an external store is created.

   ```
   * | create table user_meta1 ( userid bigint, nick varchar, gender varchar, province varchar, gender varchar,age bigint) with ( endpoint='oss-cn-hangzhou-internal.aliyuncs.com',accessid='**********',accesskey ='**********',bucket='testoss*********',objects=ARRAY['user.csv'],type='oss')
   ```

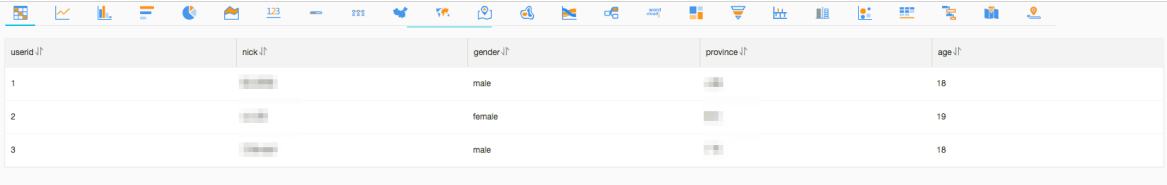
   Define the name and table schema of the external store in the SQL statement, and define the information required to access OSS objects in the WITH clause. The following table describes the parameters.

Parameter	Description
External store name	The name of the external store, which is the same as the name of the virtual external table. In this example, enter user_meta1.
Table schema	The properties of the virtual external table, including the column names and data types. In this example, enter (userid bigint, nick varchar, gender varchar, province varchar, gender varchar,age bigint).
endpoint	The internal endpoint of OSS. For more information, see Regions and endpoints.
accessid	The AccessKey ID of your account. For more information, see AccessKey pair.
accesskey	The AccessKey secret of your account. For more information, see AccessKey pair.
bucket	The OSS bucket in which the CSV object is stored.
objects	The path of the CSV object. **Note** The value of the objects parameter is an array. The array can contain multiple elements. Each element represents an OSS object.
type	The type of the external store. Set the value to oss.

5. Check whether the external store is created.

   Execute the following statement. If the table content that you define is returned, the external store is created.

```
* | select * from user_meta1
```

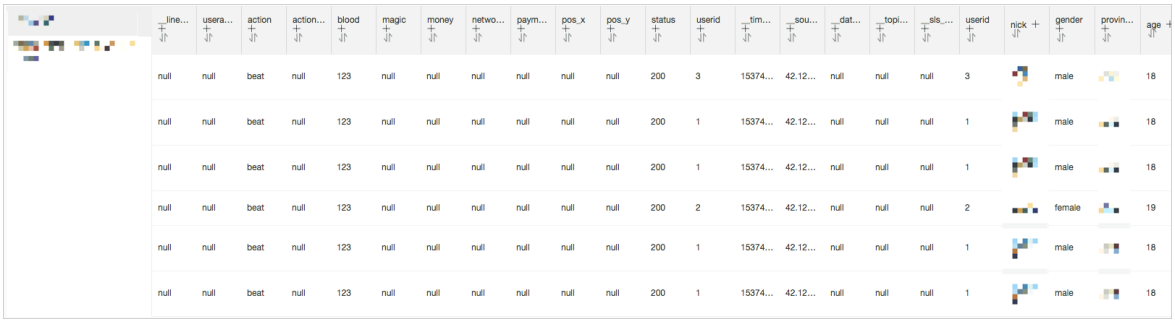


userid	nick	gender	province	age
1		male		18
2		female		19
3		male		18

6. Perform a JOIN query on Log Service and OSS.

   Execute the following statement to perform a JOIN query. A Logstore is associated with OSS objects based on the ID field in the Logstore and the userid field in the OSS objects. test_accesslog is the name of the Logstore. l is the alias of the Logstore. user_meta1 is the name of the external store that you define. You can configure the parameters based on your business requirements.

```
* | select * from test_accesslog l join user_meta1 u on l.userid = u.userid
```



_line...	usera...	action	action...	blood	magic	money	netwo...	paym...	pos_x	pos_y	status	userid	tim...	sou...	dat...	topi...	sls_...	userid	nick	gender	provin...	age
null	null	beat	null	123	null	null	null	null	null	null	200	3	15374...	42.12...	null	null	null	3		male		18
null	null	beat	null	123	null	null	null	null	null	null	200	1	15374...	42.12...	null	null	null	1		male		18
null	null	beat	null	123	null	null	null	null	null	null	200	1	15374...	42.12...	null	null	null	1		male		18
null	null	beat	null	123	null	null	null	null	null	null	200	2	15374...	42.12...	null	null	null	2		female		19
null	null	beat	null	123	null	null	null	null	null	null	200	1	15374...	42.12...	null	null	null	1		male		18
null	null	beat	null	123	null	null	null	null	null	null	200	1	15374...	42.12...	null	null	null	1		male		18

For more information about the best practices on how to associate Log Service with OSS buckets, see Associate a Logstore with an OSS external table to perform query and analysis.

12.4. Associate Log Service with a hosted CSV file

Log Service allows you to upload a CSV file from your computer to Log Service by using an SDK. This way, the CSV file is hosted on Log Service and can be associated with a Logstore of Log Service. This topic describes how to perform a JOIN query on a CSV file that is hosted on Log Service and data in a Logstore of Log Service.

Prerequisites

- Logs are collected. For more information, see Log collection methods.
- Indexes are created. For more information, see Configure indexes.
- A CSV file is created.
- Log Service SDK for Python is installed. For more information, see Install Log Service SDK for Python.

  aliyun-log-python-sdk V0.7.3 and later are supported. You can use the **pip install aliyun-log-python-sdk -U** command to upgrade the SDK.

Limits

- Only one CSV file can be associated at a time.
- You can associate Log Service with a CSV file that contains no more than 50 MB of data. The CSV file is uploaded to Log Service after it is compressed by using the SDK. The size of the file after compression must be

less than 9.9 MB.

Sample data

The Logstore stores the logon operations of a user, and the CSV file records the basic information of the user, such as the gender and age. After you associate the Logstore with the CSV file, you can analyze the metrics for user properties.

- Logstore

```
userid:100001
action:login
__time__:1637737306
```

- CSV file

userid	nick	gender	province	age
100001	User_A	male	Liaoning	24
100002	User_B	male	Beijing	23
100003	User_C	female	Zhejiang	22
100004	User_D	female	Jiangxi	21
100005	User_E	male	Guangxi	20

Procedure

1. Use Log Service SDK for Python to create an external store.

   For more information about Log Service SDK for Python, see Overview.

```
from aliyun.log import *
endpoint='cn-shanghai.log.aliyuncs.com'
accessKeyId='test-project'
accessKey='TAI****YDw'
project='lr****VM'
ext_logstore='user_meta'
csv_file='./user.csv'
client = LogClient(endpoint, accessKeyId, accessKey)
res = client.create_external_store(project,
    ExternalStoreCsvConfig(ext_logstore, csv_file,
        [
            {"name" : "userid", "type" : "bigint"},
            {"name" : "nick", "type" : "varchar"},
            {"name" : "gender", "type" : "varchar"},
            {"name" : "province", "type" : "varchar"},
            null
        ]))
res.log_print()
```

Parameter	Description
endpoint	The Log Service endpoint. For more information, see Endpoints.
accessKeyId	The AccessKey ID of your Alibaba Cloud account. For more information, see AccessKey pair. **Warning** We recommend that you use the AccessKey pair of a RAM user to call API operations. This reduces the risks caused by the leak of an AccessKey pair.

Parameter	Description
accessKey	The AccessKey secret of your Alibaba Cloud account. For more information, see AccessKey pair.
project	The project to which the Logstore belongs.
ext_logstore	The name of the external store, which is the same as the name of the virtual external table. The name must meet the following requirements: ○ The name can contain only lowercase letters, digits, hyphens (-), and underscores (_). ○ The name must start and end with a lowercase letter or a digit. ○ The name must be 3 to 63 characters in length.
csv_file	The path and name of the CSV file.
Table schema	The properties of the virtual external table, including the column names and data types. The following table schema is an example. You can replace the table schema based on your business requirements. `[ {"name" : "userid", "type" : "bigint"}, {"name" : "nick", "type" : "varchar"}, {"name" : "gender", "type" : "varchar"}, {"name" : "province", "type" : "varchar"}, null ]`

2.
3.
4.
5. Execute the following statement to check whether the external store is created.

   In the following statement, `user_meta` specifies the name of the external store. Replace the value with the name that you specify when you create the external store.

   ```
   * | SELECT * FROM user_meta
   ```

If the content of the CSV file is returned, the external store is created.

userid	nick	gender	province	age
100001	User_A	male	Liaoning	24
100002	User_B	male	Beijing	23
100003	User_C	female	Zhejiang	22
100004	User_D	female	Jiangxi	21
100005	User_E	male	Guangxi	20

6. Execute the following statement to perform a JOIN query on the Logstore and the CSV file.

   The Logstore is associated with the CSV file based on the userid field in the Logstore and the userid field in the CSV file. website_log is the name of the Logstore. user_meta is the name of the external store that you create. You can configure the parameters based on your business requirements.

```
* | SELECT * FROM website_log JOIN user_meta ON website_log.userid = user_meta.userid
```

action	userid	__time__	userid	nick	gender	province	age
login	100003	1637738249	100003	User_C	female	Zhejiang	22
login	100004	1637738249	100004	User_D	female	Jiangxi	21
login	100005	1637738249	100005	User_E	male	Guangxi	20
login	100002	1637738249	100002	User_B	male	Beijing	23
login	100004	1637738249	100004	User_D	female	Jiangxi	21
login	100003	1637738249	100003	User_C	female	Zhejiang	22
login	100001	1637738249	100001	User_A	male	Liaoning	24

13.Best practices

13.1. Query and analyze website logs

This topic describes how to query and analyze website logs in the Log Service console.

Prerequisites

Website access logs are collected. For more information, see Collect logs in full regex mode.

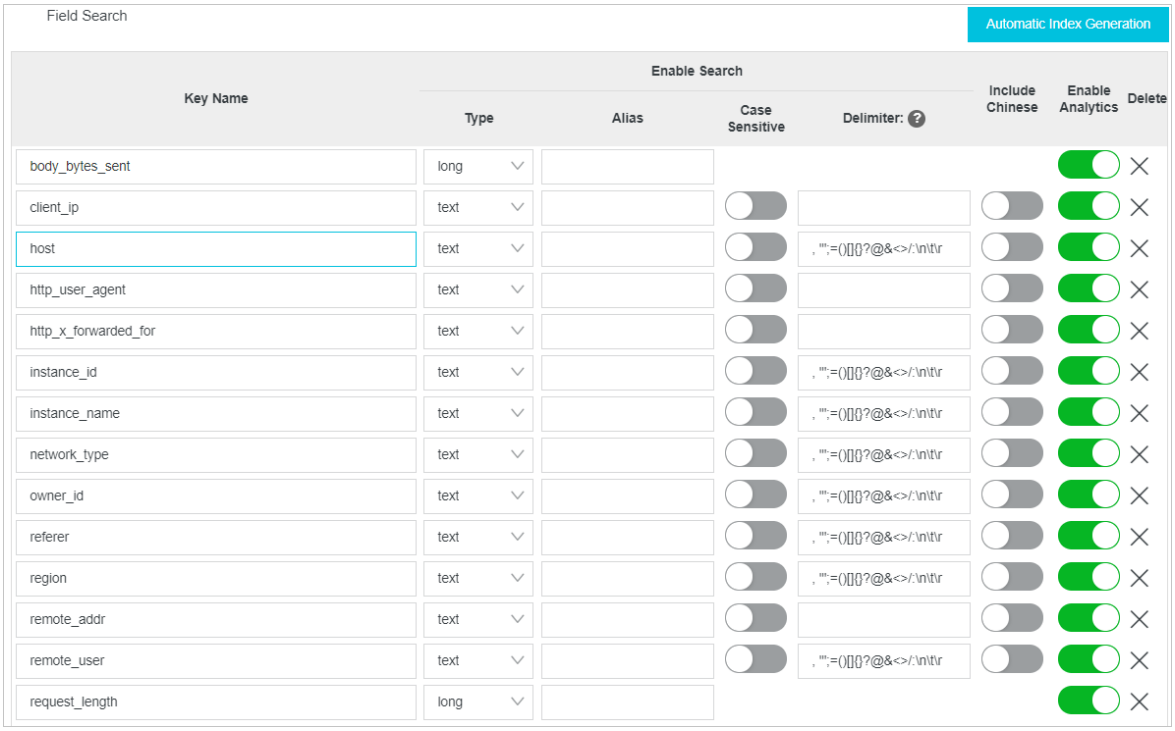
Step 1: Configure indexes

1.
2.
3.
4. On the Search & Analysis page of the Logstore, choose **Index Attributes > Attributes**.

   If the indexing feature is not enabled, click **Enable**.
5. Configure field indexes.

   You can configure indexes in sequence. You can also click **Automatic Index Generation**. Then, Log Service automatically configures indexes based on the first log entry in the Preview Data section.

   > **Note**
   > - The indexing feature is applicable only to the log data that is written to the current Logstore after you configure indexes. If you want to query historical data, you can use the reindexing feature. For more information, see Reindex logs for a Logstore.
   > - If you want to use the analysis feature, you must turn on the Enable Analytics switch for the related fields when you configure indexes.
   > - By default, indexes are automatically configured for some reserved fields in Log Service. For more information, see Reserved fields.



6. Click **OK**.

Step 2: Query logs

On the Search & Analysis page of the Logstore, enter a search statement in the search box and select a time range. Then, click **Search & Analyze**.

> **Note** The format of a query statement in Log Service is `Search statement | Analytic statement`. A search statement can be executed alone. However, an analytic statement must be executed together with a search statement. The analysis feature is based on search results or all data in the Logstore.

- To query the log entries that contain Chrome, execute the following search statement:

```
Chrome
```

- To query the log entries whose request duration is greater than 60 seconds, execute the following search statement:

```
request_time > 60
```

- To query the log entries whose request duration ranges from 60 seconds to 120 seconds, execute the following search statement:

```
request_time in [60 120]
```

- To query the log entries that contain successful GET requests (status code: 200 to 299), execute the following search statement:

```
request_method : GET and status in [200 299]
```

- To query the log entries whose value of the request_uri field is /request/path-2, execute the following search statement:

```
request_uri:/request/path-2/file-2
```

Step 3: Analyze logs

On the Search & Analysis page of the Logstore, enter a query statement in the search box and select a time range. Then, click **Search & Analyze**.

> **Note** By default, only 100 rows of data are returned after you execute a query statement. You can use the LIMIT clause to change the number of returned rows. For more information, see LIMIT clause.

- Calculate the page views (PVs) of a website.

  To calculate the PVs of a website, use the COUNT function in a query statement.

  ```
  * | SELECT COUNT(*) AS PV
  ```

PV
9685

- Calculate the PVs of a website by 1 minute.

  Use the date_trunc function to truncate a time by minute and use the GROUP BY clause to group analysis results by time. Then, use the COUNT function to calculate the number of PVs per minute and use the ORDER BY clause to sort the analysis results by time.

  ```
  * | SELECT COUNT(*) as PV, date_trunc('minute', __time__) as time GROUP BY time ORDER BY time
  ```

PV	time
73	2021-01-15 09:15:00.000
472	2021-01-15 09:16:00.000
693	2021-01-15 09:17:00.000
919	2021-01-15 09:18:00.000
520	2021-01-15 09:19:00.000

- Calculate the number of requests for each request method by 5 minutes.

  Use **__time__ - __time__ %300** to truncate a time by 5 minutes and use the GROUP BY clause to group analysis results by time. Then, use the COUNT function to calculate the number of requests every 5 minutes and use the ORDER BY clause to sort the analysis results by time.

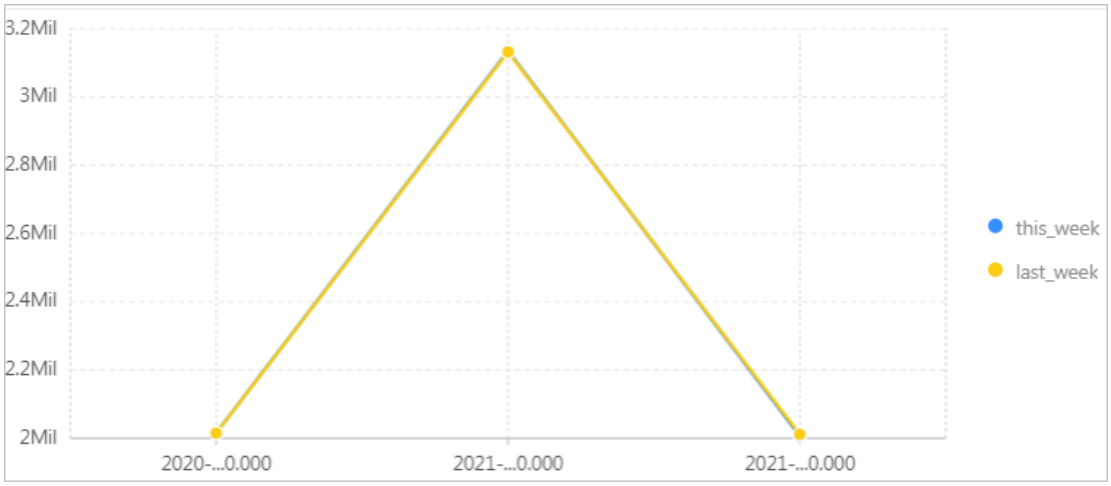
  ```
  * | SELECT request_method, COUNT(*) as count, __time__ - __time__ %300 as time GROUP BY time, requ
  est_method ORDER BY time
  ```

request_method	count	time
PUT	242	1610673300
DELETE	101	1610673300
POST	231	1610673300
GET	778	1610673300
HEAD	4	1610673300

- Compare the number of PVs of the current week with the number of PVs of the last week.

Use the COUNT function to calculate the total number of PVs. Then, use the ts_compare function to obtain the ratio of the PVs of the current week to the PVs of the last week. website_log in the following query statement is the Logstore name:

```
* | SELECT diff[1] as this_week, diff[2] as last_week, time FROM (SELECT ts_compare(pv, 604800) as diff, time FROM (SELECT COUNT(*) as pv, date_trunc('week', __time__) as time FROM website_log GROUP BY time ORDER BY time) GROUP BY time)
```



- Collect the distribution statistics of client IP addresses.

  Use the ip_to_province function to obtain the province to which an IP address belongs, and use the GROUP BY clause to group analysis results by province. Then, use the COUNT function to calculate the number of occurrences of each IP address, and use the ORDER BY clause to sort the analysis results by the number of occurrences.

  ```
  * | SELECT COUNT(*) as count, ip_to_province(client_ip) as address GROUP BY address ORDER BY count DESC
  ```

count	address
451	
447	
433	
425	

- Calculate the top 10 accessed request URIs.

  Use the GROUP BY clause to group analysis results by request URI. Use the COUNT function to calculate the number of access requests for each URI. Then, use the ORDER BY clause to sort the analysis results by the number of access requests.

  ```
  * | SELECT COUNT(*) as PV, request_uri as PATH GROUP BY PATH ORDER BY PV DESC LIMIT 10
  ```

PV	PATH
283	/request/path-0/file-9
277	/request/path-0/file-6
263	/request/path-3/file-1
262	/request/path-0/file-1

- Query the log entries whose value of the request_uri field ends with %file-7.

  > **Note** In query statements, the wildcard characters asterisk (*) and question mark (?) are used for fuzzy searches. The wildcard characters must be used in the middle or at the end of a word. If you want to query fields that end with a specific character, you can use the LIKE operator in an analytic statement.

  ```
  * | select * from website_log where request_uri like '%file-7'
  ```

  website_log in the preceding query statement is the Logstore name.

request_method	request_time	request_uri	scheme	server_protocol	slbid	status
POST	67	/request/path-2/file-7	https	HTTP/2.0	slb-01	200
GET	41	/request/path-0/file-7	https	HTTP/2.0	slb-02	305

- Calculate the statistics of request URIs that are accessed.

  Use the regexp_extract function to extract the file part from the request_uri field. Then, use the COUNT function to calculate the number of access requests for each URI.

  ```
  * | SELECT regexp_extract(request_uri, '.*\/(file.*)', 1) file, count(*) as count group by file
  ```

file	count
file-5	17127

- Query the log entries whose value of the request_uri field contains %abc%.

  ```
  * | SELECT * where request_uri like '%/%abc/%%' escape '/'
  ```

__line__	http_user_agent	region	request_uri	scheme	server_protocol	time
null	Mozilla/5.0	cn-shanghai	/request/path-1/file-9?%abc%qereqwr	https	HTTP/2.0	upstnse_que

Sample logs

```
__tag__:__client_ip__:192.0.2.0
__tag__:__receive_time__:1609985755
__source__:198.51.100.0
__topic__:website_access_log
body_bytes_sent:4512
client_ip:198.51.100.10
host:example.com
http_host:example.com
http_user_agent:Mozilla/5.0 (Macintosh; U; PPC Mac OS X 10_5_8; ja-jp) AppleWebKit/533.20.25 (KHTML,
like Gecko) Version/5.0.4 Safari/533.20.27
http_x_forwarded_for:198.51.100.1
instance_id:i-02
instance_name:instance-01
network_type:vlan
owner_id:%abc%-01
referer:example.com
region:cn-shanghai
remote_addr:203.0.113.0
remote_user:neb
request_length:4103
request_method:POST
request_time:69
request_uri:/request/path-1/file-0
scheme:https
server_protocol:HTTP/2.0
slbid:slb-02
status:200
time_local:07/Jan/2021:02:15:53
upstream_addr:203.0.113.10
upstream_response_time:43
upstream_status:200
user_agent:Mozilla/5.0 (X11; Linux i686) AppleWebKit/534.33 (KHTML, like Gecko) Ubuntu/9.10 Chromium/
13.0.752.0 Chrome/13.0.752.0 Safari/534.33
vip_addr:192.0.2.2
vpc_id:3db327b1****82df19818a72
```

13.2. Query and analyze JSON logs

This topic describes how to query and analyze JSON logs in the Log Service console.

Prerequisites

JSON logs are collected. For more information, see 使用极简模式采集日志.

Usage notes

When you query and analyze JSON logs, make sure that the following requirements are met:

- The format of a query statement in Log Service is `Search statement | Analytic statement`. In an analytic statement, you must use double quotation marks ("") to enclose a field name and use single quotation marks ('') to enclose a string.
- You must add all parent paths to a specified field in the KEY1.KEY2.KEY3 format, for example, concent.request.request_length.
- Log Service allows you to query and analyze leaf nodes in JSON objects. However, you cannot query or analyze child nodes that contain leaf nodes.
- You cannot query or analyze fields whose values are JSON arrays. In addition, you cannot query or analyze the fields in a JSON array.

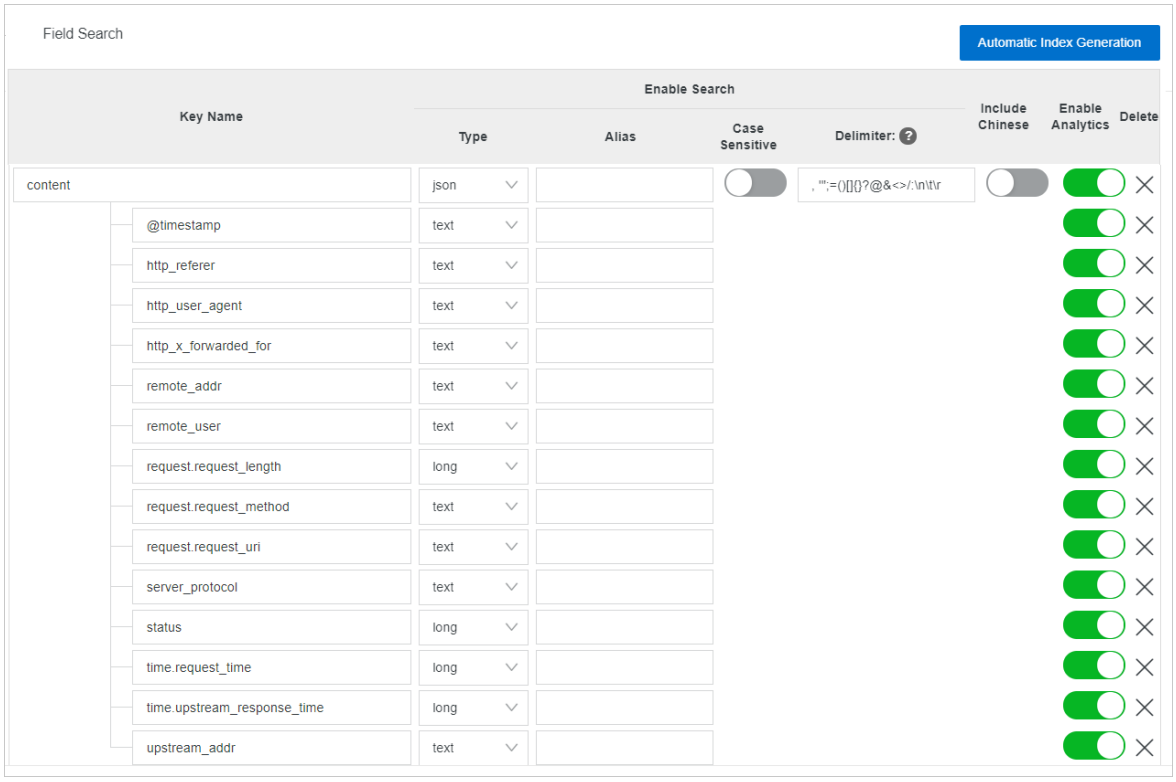
Step 1: Configure indexes

1.
2. In the **Projects** section, click the project in which you want to query and analyze logs.
3. Choose **Log Storage > Logstores**. On the Logstores tab, click the Logstore where logs are stored.
4. On the Search & Analysis page of the Logstore, choose **Index Attributes > Attributes**.

   If the indexing feature is not enabled, click **Enable**.
5. Configure field indexes.

   You can configure indexes one by one. You can also click **Automatic Index Generation**. Then, Log Service automatically configures indexes based on the first log entry in the Preview Data section.

   > **Note**
   > - If you want to use the analysis feature, you must turn on the Enable Analytics switch for the related fields when you configure indexes. For more information, see Configure indexes.
   > - By default, indexes are automatically configured for some reserved fields in Log Service. For more information, see Reserved fields.
   > - Log Service allows you to configure indexes for leaf nodes in JSON objects. However, you cannot configure indexes for child nodes that contain leaf nodes. For example, you can create an index for the request_time field, but you cannot create an index for the time field.
   > - You cannot configure indexes for fields whose values are JSON arrays. In addition, you cannot configure indexes for the fields in a JSON array. For example, the value of the body_bytes_sent field is a JSON array. In this case, you cannot create an index for the field.
   > - When you configure indexes for the fields in JSON objects, you must add the related parent path in the KEY1.KEY2 format. Example: time.request_time.



6. Click **OK**.

> **Note** The indexing feature is applicable only to the log data that is written to the current Logstore after you configure indexes. If you want to query historical data, you can use the reindexing feature. For more information, see Reindex logs for a Logstore.

Step 2: Query logs

On the Search & Analysis page of the Logstore, enter a search statement in the search box and select a time range. Then, click **Search & Analyze**.

- Query the log entries whose response status code is 200.

```
content.status:200
```

- Query the log entries whose request length is greater than 70.

```
content.request.request_length > 70
```

- Query the log entries that contain GET requests.

```
content.request.request_method:GET
```

Step 3: Analyze logs

On the Search & Analysis page of the Logstore, enter a query statement in the search box and select a time range. Then, click **Search & Analyze**.

- Calculate the number of log entries for each request status.

```
* | SELECT "content.status", COUNT(*) AS PV GROUP BY "content.status"
```

content.status	PV
200	45793
null	11137

- Calculate the number of requests for each request duration and sort analysis results by request duration in ascending order.

```
* | SELECT "content.time.request_time", COUNT(*) AS count GROUP BY "content.time.request_time" ORD
ER BY "content.time.request_time"
```

content.time.request_time	count
10.0	145
11.0	123
12.0	113

- Calculate the average request duration for each request method.

```
* | SELECT avg("content.time.request_time") AS avg_time,"content.request.request_method"  GROUP BY
"content.request.request_method"
```

avg_time	content.request.request_method
45	GET
11	PUT

Sample logs

The following figure shows sample JSON logs.

13.3. Associate a Logstore with a MySQL database to perform query and analysis

This topic describes how to associate a Logstore with a MySQL database to perform query and analysis. In this topic, the logs of a gaming company are used as an example.

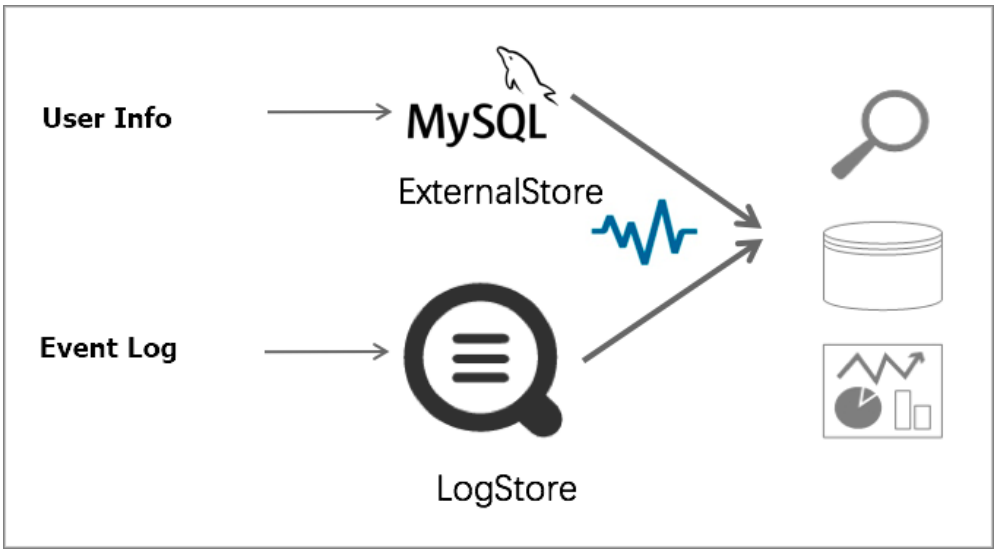
Prerequisites

- Logs are collected to a Logstore. For more information, see Data collection overview.
- Indexes are created for the log fields. For more information, see Configure indexes.
- A MySQL database is available. For more information, see Create databases and accounts for an ApsaraDB RDS for MySQL instance.

Context

Company A is a gaming company that has two types of data: user game logs and user metadata. Log Service can collect user game logs in real time. A user game log contains event information such as the operation, targets, health points (HP), magic points (MP), network, payment method, click location, status code, and user ID. User metadata includes user information such as the gender, registration time, and region. In most cases, user metadata is stored in a database because metadata cannot be displayed in logs. Company A wants to perform association analysis on the user game logs and user metadata to obtain an optimal operations plan.

The query and analysis engine of Log Service allows you to associate Logstores with external stores to perform query and analysis. External stores include MySQL databases and Object Storage Service (OSS) buckets. To analyze the metrics that are related to user properties, you can use the SQL JOIN syntax to associate the user game logs with the user metadata. You can also write analysis results to external stores to process the results.



Procedure

1. Create a user property table in the MySQL database.

   Create a table named chiji_user to store user IDs, usernames, genders, ages, account balance, registration time, and registration regions.

   ```
   CREATE TABLE `chiji_user` (
     `uid` int(11) NOT NULL DEFAULT '0',
     `user_nick` text,
     `gender` tinyint(1) DEFAULT NULL,
     `age` int(11) DEFAULT NULL,
     `register_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
     `balance` float DEFAULT NULL,
     `region` text, PRIMARY KEY (`uid`)
   ) ENGINE=InnoDB DEFAULT CHARSET=utf8
   ```

2. Create a whitelist for the MySQL database.
3. Create an external store.
   i. Install the Log Service CLI. For more information, see CLI Overview.
   ii. Create a configuration file named */root/config.json*.
   iii. Add the following script to the */root/config.json* file. Replace the parameter values based on your business requirements.

```
{
"externalStoreName":"storename",
"storeType":"rds-vpc",
"parameter":
   {
   "region":"cn-qingdao",
   "vpc-id":"vpc-m5eq4irc1pucp*******",
   "instance-id":"i-m5eeo2whsn*******",
   "host":"localhost",
   "port":"3306",
   "username":"root",
   "password":"****",
   "db":"scmc",
   "table":"join_meta"
   }
}
```

Parameter	Description
externalStoreName	The name of the external store. The name must be in lowercase.
storeType	The type of the data source. Set the value to rds-vpc.
region	The region. In the following descriptions, an ApsaraDB RDS for MySQL instance is referred to as an RDS instance. ■ If you use a MySQL database created on an RDS instance, set region to the region where the RDS instance resides. ■ If you use a self-managed MySQL database hosted on a VPC-type ECS instance, set region to the region where the ECS instance resides. ■ If you use a self-managed MySQL database created in other scenarios, set region to an empty string. Format: **"region": ""**.
vpc-id	The ID of the VPC. ■ If you use a MySQL database created on a VPC-type RDS instance, set vpc-id to the ID of the VPC to which the RDS instance belongs. ■ If you use a self-managed MySQL database hosted on a VPC-type ECS instance, set vpc-id to the ID of the VPC to which the ECS instance belongs. ■ If you use a MySQL database created on a classic network-type RDS instance or if you use a self-managed MySQL database created in other scenarios, set vpc-id to an empty string. Format: **"vpc-id": ""**.
instance-id	The ID of the instance. ■ If you use a MySQL database created on an RDS instance, set instance-id to the value of the VpcCloudInstanceId parameter that is specified for the RDS instance. You can call the DescribeDBInstanceAttribute operation to obtain the value of the VpcCloudInstanceId parameter. ■ If you use a self-managed MySQL database hosted on a VPC-type ECS instance, set instance-id to the ID of the ECS instance. ■ If you use a self-managed MySQL database created in other scenarios, set instance-id to an empty string. Format: **"instance-id": ""**.

Parameter	Description
host	The address of your MySQL database. ■ If you use a MySQL database created on a VPC-type RDS instance, set host to an internal endpoint of the RDS instance. ■ If you use a self-managed MySQL database hosted on a VPC-type ECS instance, set host to the private IP address of the ECS instance. ■ If you use a self-managed MySQL database created in other scenarios, set host to a host address of the database. Make sure that the host address is accessible.
port	The port number. ■ If you use a MySQL database created on an RDS instance, set port to the port of the RDS instance. ■ If you use a self-managed MySQL database hosted on a VPC-type ECS instance, set port to the MySQL service port of the ECS instance. ■ If you use a self-managed MySQL database created in other scenarios, set port to the MySQL service port.
username	The username of the account that you use to log on to your MySQL database.
password	The password of the account that you use to log on to your MySQL database.
db	The name of your MySQL database.
table	The name of the table that you want to use in your MySQL database.

iv. Create an external store.

Replace the value of project_name with the name of the project that you want to use.

```
aliyunlog log create_external_store --project_name="log-rds-demo" --config="file:///root/config.json"
```

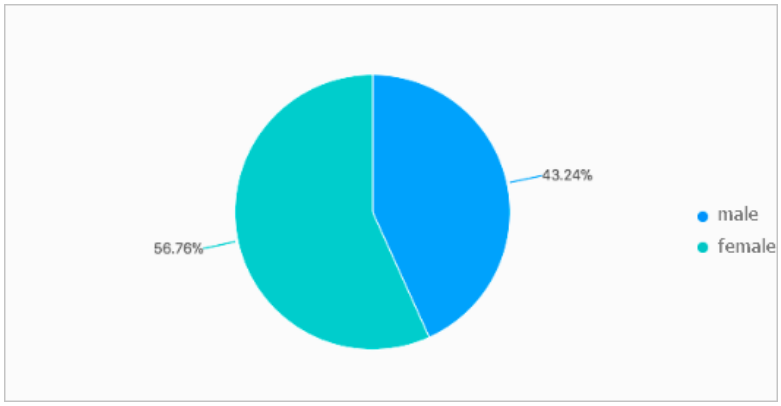
4. Use the SQL JOIN syntax to perform a JOIN query.
   i. Log on to the Log Service console.
   ii. In the **Projects** section, click the project in which you want to query and analyze logs.
   iii. On the **Log Storage > Logstores** tab, click the Logstore where logs are stored.

iv. Execute a query statement.

To perform a JOIN query, use the userid field in the Logstore and the uid field in the database table.

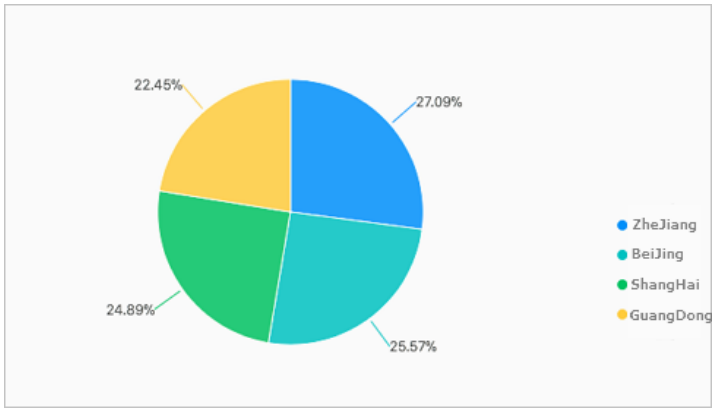
- Analyze the distribution of active users by gender.

```
* | select case gender when 1 then 'Male' else 'Female' end as gender , count(1) as pv fro
m log l join chiji_user u on l.userid = u.uid group by gender order by pv desc
```



- Analyze the activity levels of users in different regions.

```
* | select region , count(1) as pv from log l join chiji_user u on l.userid = u.uid group
by region order by pv desc
```



- Analyze the consumption trends of users of different genders.

```
* | select case gender when 1 then 'Male' else 'Female' end as gender , sum(money) as mone
y from log l join chiji_user u on l.userid = u.uid group by gender order by money desc
```

5. Save the query and analysis results to the MySQL database.

i. Create a data table named report in the MySQL database to store the number of page views (PVs) per minute.

```
CREATE TABLE `report` (
  `minute` bigint(20) DEFAULT NULL,
  `pv` bigint(20) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

ii. Create an external store for the report table. For more information, see Step .

iii. On the search and analysis page of the Logstore, execute the following query statement to save the results to the report table:

```
* | insert into report select __time__- __time__ % 300 as min, count(1) as pv group by min
```

After the results are saved, you can view the results in the MySQL database.

```
mysql> select * from report;
+------------+-------+
| minute     | pv    |
+------------+-------+
| 1526448600 |  3000 |
| 1526448540 |  9900 |
| 1526448780 |  3100 |
| 1526448480 |  5400 |
| 1526448720 |  3000 |
| 1526448960 |  3000 |
| 1526448900 |  3000 |
| 1526449080 |  3000 |
| 1526449140 |  3000 |
| 1526448660 |  2900 |
| 1526449260 |  3000 |
```

13.4. Associate a Logstore with an OSS external table to perform query and analysis

In some scenarios, you may need to use external tables to query and analyze logs. This topic describes how to associate a Logstore with an Object Storage Service (OSS) external table to perform query and analysis.

Prerequisites

- Logs are collected. For more information, see Data collection overview.
- Indexes are created. For more information, see Configure indexes.
- An OSS bucket is created. For more information, see Create buckets.

Context

Company A is an electronic payment company. Company A wants to analyze the impact of user ages, geographic locations, and genders on payment preferences. Company A has collected payment behavior logs by using Log Service and stored user property information in an OSS bucket. The payment behavior logs contain bills and payment methods. The user property information includes the geographic location, age, and gender information about a user. The query and analysis engine of Log Service allows you to associate Logstores with external stores to perform query and analysis. External stores include MySQL databases and OSS buckets. To analyze the metrics that are related to user properties, you can use the SQL JOIN syntax to associate the payment behavior logs with the user property information.

If you associate Log Service with OSS buckets to perform query and analysis, you can receive the following benefits:

- Cost-effectiveness: If you store infrequently updated data in OSS buckets, the data can be read over an internal network. In this case, you need only to pay for the storage service, and you are not charged for Internet traffic.
- Reduced O&M workload: You can perform lightweight association analysis without the need to store all data in one storage system.
- High efficiency: You can use SQL statements to analyze data and view the analysis results within seconds. You can also create charts based on analysis results that are commonly queried. Then, you can click the charts to

view the analysis results.

Procedure

1. Create a CSV file and upload it to the OSS bucket.
   i. Create a file named *user.csv*.

   ```
   userid,nick,gender,province,age
   1,User A,male,Shanghai,18
   2,User B,female,Zhejiang,19
   3,User C,male,Guangdong,18
   ```

   ii. Upload the *user.csv* file to the OSS bucket. For more information, see Upload objects.
2.
3.
4.
5. On the page that appears, enter a query statement in the search box and click **Search & Analyze**.

   Execute the following SQL statement to create a virtual external table named user_meta1 and map the table to the OSS object user.csv. If **result** in the output is **true**, the SQL statement is successfully executed, and an external store is created.

   ```
   * | create table user_meta1 ( userid bigint, nick varchar, gender varchar, province varchar, age bigint) with ( endpoint='oss-cn-hangzhou.aliyuncs.com',accessid='LTAI5t8y9c113M7V****',accesskey='Y45H7bqvvgapWZR****',bucket='testoss',objects=ARRAY['user.csv'],type='oss')
   ```

   Define the name and table schema of the external store in the SQL statement, and define the information required to access OSS objects in the WITH clause. The following table describes the parameters.

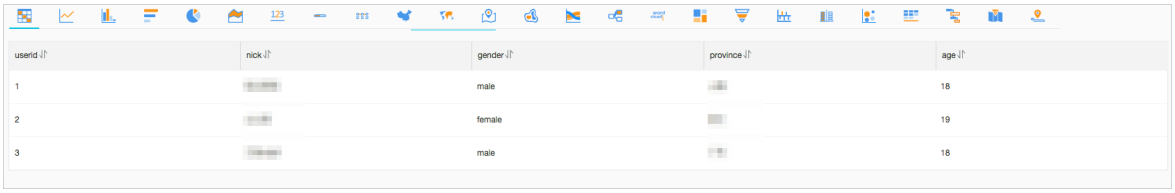
Parameter	Description	Example
External store name	The name of the external store, which is the same as the name of the virtual external table.	user_meta1
Table schema	The properties of the virtual external table, including the column names and data types. In this example, enter (userid bigint, nick varchar, gender varchar, province varchar, age bigint).	(userid bigint, nick varchar, gender varchar, province varchar, age bigint)
endpoint	The internal endpoint of OSS. For more information, see Regions and endpoints.	oss-cn-hangzhou.aliyuncs.com
accessid	The AccessKey ID of your account. For more information, see AccessKey pair.	LTAI5t8y9c113M7V****
accesskey	The AccessKey secret of your account. For more information, see AccessKey pair.	Y45H7bqvvgapWZR****
bucket	The OSS bucket in which the CSV object is stored.	testoss

Parameter	Description	Example
objects	The path of the CSV object. **Note** The value of the objects parameter is an array. The array can contain multiple elements. Each element represents an OSS object.	user.csv
type	The type of the external store. Set the value to oss.	oss

6. Check whether the external store is created.

   Execute the following statement. If the table content that you define is returned, the external store is created. user_meta1 is the name of the external store. You can configure this parameter based on your business requirements.

   ```
   * | select * from user_meta1
   ```

7. Perform a JOIN query on Log Service and OSS.

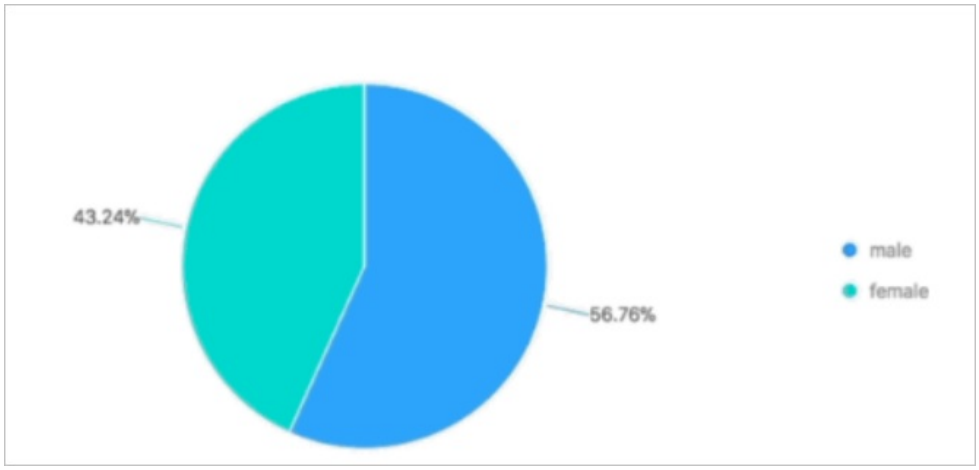
   Execute the following statement to perform a JOIN query. A Logstore is associated with OSS objects based on the ID field in the Logstore and the userid field in the OSS objects. test_accesslog is the name of the Logstore. l is the alias of the Logstore. user_meta1 is the name of the external store that you define. You can configure the parameters based on your business requirements.

   ```
   * | select * from test_accesslog l join user_meta1 u on l.userid = u.userid
   ```

   Examples:

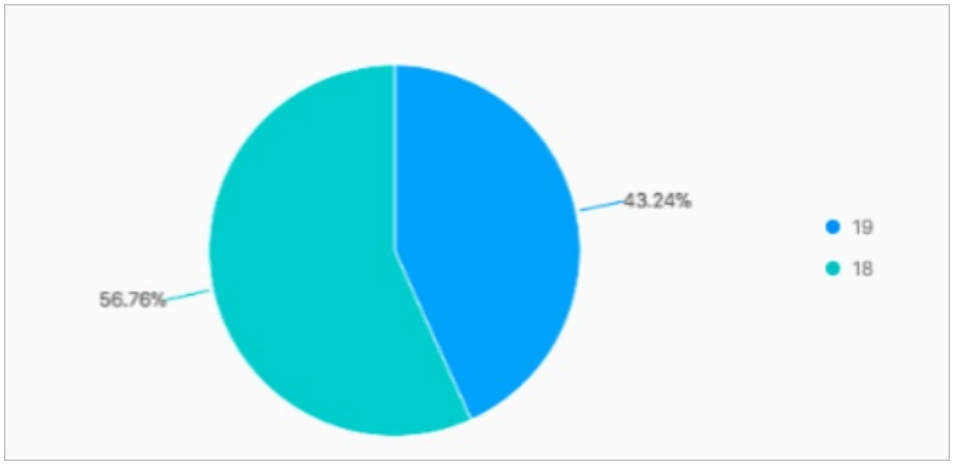
   - Analyze the access requests from users of different genders.

     ```
     * | select u.gender, count(1) from test_accesslog l join user_meta1 u on l.userid = u.userid g
     roup by u.gender
     ```

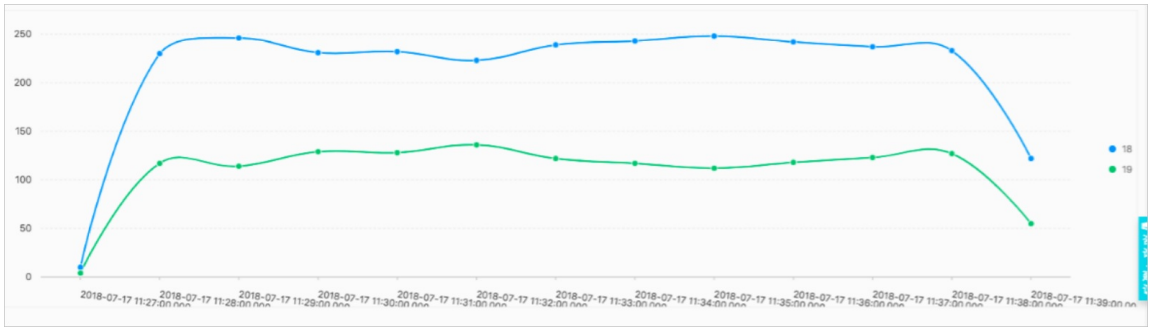
   - Analyze the access requests from users of different ages.

```
* | select u.age, count(1) from test_accesslog l join user_meta1 u on l.userid = u.userid grou
p by u.age
```



- Analyze the access trends of users of different ages in different time ranges.

```
* | select date_trunc('minute',__time__) as minute, count(1) ,u.age from test_accesslog l join
user_meta1 u on l.userid = u.userid group by u.age,minute
```



13.5. Collect and analyze NGINX monitoring logs

NGINX provides a built-in status page that allows you to monitor the status of NGINX. This topic describes how to use the Logtail feature of Log Service to collect NGINX status information. The topic also describes how to query and analyze the collected status information, and customize alerts to monitor your NGINX cluster.

Prerequisites

Logtail is installed on the server that you use to collect MySQL binary logs. For more information, see Install Logtail on a Linux server or 安装Logtail（Windows系统）.

> **Note** Servers that run Linux support Logtail 0.16.0 or later. Servers that run Windows support Logtail 1.0.0.8 or later.

Step 1: Prepare the environment

Perform the following steps to enable the NGINX status page:

1. Run the following command to check whether NGINX supports the status feature. For more information, see Module ngx_http_stub_status_module.

```
nginx -V 2>&1 | grep -o with-http_stub_status_module
with-http_stub_status_module
```

If the message `with-http_stub_status_module` is returned, NGINX supports the status feature.

2. Configure the NGINX status feature.

   Enable the status feature in the NGINX configuration file. This file is stored in the /etc/nginx/nginx.conf directory. Use the following example to configure the status feature. For more information, see Enable Nginx Status Page.

   > **Note** In the preceding example, allow 10.10.XX.XX indicates that only the server whose IP address is 10.10.XX.XX is allowed to access the NGINX status page.

   ```
   location /private/nginx_status {
     stub_status on;
     access_log   off;
     allow 10.10.XX.XX;
     deny all;
   }
   ```

3. Run the following command to verify whether the server on which Logtail is installed can access the NGINX status page:

   ```
   $curl http://10.10.XX.XX/private/nginx_status
   ```

   If the following message is returned, the NGINX status page is enabled:

   ```
   Active connections: 1
   server accepts handled requests
   2507455 2507455 2512972
   Reading: 0 Writing: 1 Waiting: 0
   ```

Step 2: Collect NGINX monitoring logs

1.
2. In the **Import Data** section, select **Custom Data Plug-in**.
3.
4.
5.
6. In the **Specify Data Source** step, set the **Config Name** and **Plug-in Config** parameters.
   - inputs: Required. The Logtail configurations for log collection.

     > **Note** You can configure only one type of data source in the inputs field.

   - processors: Optional. The Logtail configurations for data processing. You can configure one or more processing methods in the processors field. For more information, see Overview.

```
{
"inputs": [
 {
      "type": "metric_http",
      "detail": {
          "IntervalMs": 60000,
          "Addresses": [
              "http://10.10.XX.XX/private/nginx_status",
              "http://10.10.XX.XX/private/nginx_status",
              "http://10.10.XX.XX/private/nginx_status"
          ],
          "IncludeBody": true
      }
 }
],
"processors": [
 {
      "type": "processor_regex",
      "detail": {
          "SourceKey": "content",
          "Regex": "Active connections: (\\d+)\\s+server accepts handled requests\\s+(\\d+)\\s+(\
\d+)\\s+(\\d+)\\s+Reading: (\\d+) Writing: (\\d+) Waiting: (\\d+)[\\s\\S]*",
          "Keys": [
              "connection",
              "accepts",
              "handled",
              "requests",
              "reading",
              "writing",
              "waiting"
          ],
          "FullMatch": true,
          "NoKeyError": true,
          "NoMatchError": true,
          "KeepSource": false
      }
 }
]
}
```

The following table describes the required parameters.

Parameter	Type	Required	Description
type	string	Yes	The type of the data source. Set the value to metric_http.
IntervalMs	int	Yes	The interval between two successive requests. Unit: milliseconds.
Addresses	String	Yes	The list of URLs that you want to monitor.
IncludeBody	bool	No	Specifies whether to collect the request body. Default value: false. If you set the value to true, the content of the request body is stored in the field named content.

One minute after the collection configurations are complete, you can view the collected status data, as shown in the following example. By default, Log Service provides dashboards to display NGINX monitoring data.

```
_address_:http://10.10.XX.XX/private/nginx_status
_http_response_code_:200
_method_:GET
_response_time_ms_:1.83716261897
_result_:success
accepts:33591200
connection:450
handled:33599550
reading:626
requests:39149290
waiting:68
writing:145
```

Step 3: Query and analyze NGINX monitoring logs

1.
2.
3.
4. In the upper-right corner of the page, click **15 Minutes (Relative)** to set a time range for the query.

   You can select a relative time or time frame, or customize a time range.

   > **Note** The query results may contain logs that are generated one minute earlier or later than the specified time range.

5. Enter a query statement in the search box, and then click **Search & Analyze**.

   For more information, see Log analysis overview and Configure an alert in Log Service.

   - Query logs
     - To query the status of the server whose IP address is 10.10.0.0, execute the following statement:

       ```
       _address_ : 10.10.0.0
       ```

     - To query the requests whose response time is greater than 100 ms, execute the following statement:

       ```
       _response_time_ms_ > 100
       ```

     - To query the requests whose HTTP status code is not 200, execute the following statement:

       ```
       not _http_response_code_ : 200
       ```

   - Analyze logs
     - To query the average number of waiting connections, reading connections, writing connections, and connections every 5 minutes, execute the following statement:

       ```
       *| select  avg(waiting) as waiting, avg(reading)  as reading,  avg(writing)  as writing,  av
       g(connection)  as connection,  from_unixtime( __time__ - __time__ % 300) as time group by __
       time__ - __time__ % 300 order by time limit 1440
       ```

     - To query the top 10 servers with the largest number of waiting connections, execute the following statement:

       ```
       *| select  max(waiting) as max_waiting, address, from_unixtime(max(__time__)) as time group
       by address order by max_waiting desc limit 10
       ```

- To query the total number of requests and failed requests, execute the following statement:

```
* | select  count(distinct(address)) as total
```

```
not _result_ : success | select  count(distinct(address))
```

- To query the 10 most recent failed requests, execute the following statement:

```
not _result_ : success | select _address_ as address, from_unixtime(__time__) as time  order by __time__ desc limit 10
```

- To query the total number of requests handled every 5 minutes, execute the following statement:

```
*| select  avg(handled) * count(distinct(address)) as total_handled, avg(requests) * count(distinct(address)) as total_requests,  from_unixtime( __time__ - __time__ % 300) as time group by __time__ - __time__ % 300 order by time limit 1440
```

- To query the average latency of requests every 5 minutes, execute the following statement:

```
*| select  avg(_response_time_ms_) as avg_delay,  from_unixtime( __time__ - __time__ % 300) as time group by __time__ - __time__ % 300 order by time limit 1440
```

- To query the number of valid and invalid requests, execute the following statement:

```
not _http_response_code_ : 200  | select  count(1)
```

```
_http_response_code_ : 200  | select  count(1)
```

13.6. Collect and analyze NGINX access logs

Log service allows you to collect and analyze NGINX access logs. This topic describes how to monitor, analyze, diagnose, and optimize access to a website.

Prerequisites

Log data is collected. For more information, see Collect logs in NGINX mode.

The indexing feature is enabled and configured. For more information, see Configure indexes.

Context

NGINX is a free, open-source, and high-performance HTTP server that you can use to build and host websites. NGINX access logs can be collected and analyzed. In traditional methods such as CNZZ, a JavaScript script is inserted into the frontend page of a website and is triggered when a user visits the website. However, this method can record only access requests. Stream computing, offline computing, and offline analysis can also be used to analyze NGINX access logs. However, those methods require a dedicated environment and it can be difficult to balance time efficiency and flexibility during log analysis.

In the Log Service console, you can create a collection configuration to collect NGINX access logs by using the data import wizard. Then, Log Service creates indexes and an NGINX dashboard to help you collect and analyze NGINX access logs. The dashboard displays metrics such as Distribution of IP Addresses, HTTP Status Codes, Request Methods, page view (PV) and unique visitor (UV) Statistics, Inbound and Outbound Traffic, User Agents, Top 10 Request URLs, Top 10 URIs by Number of Requests, and Top 10 URIs by Request Latency. You can use query statements to analyze the access latency of your website and optimize the performance of your website at the earliest opportunity. You can create alerts to track performance issues, server errors, and traffic changes. If the trigger conditions of an alert are met, alert notifications are sent to the specified recipients.

Analyze access to a website

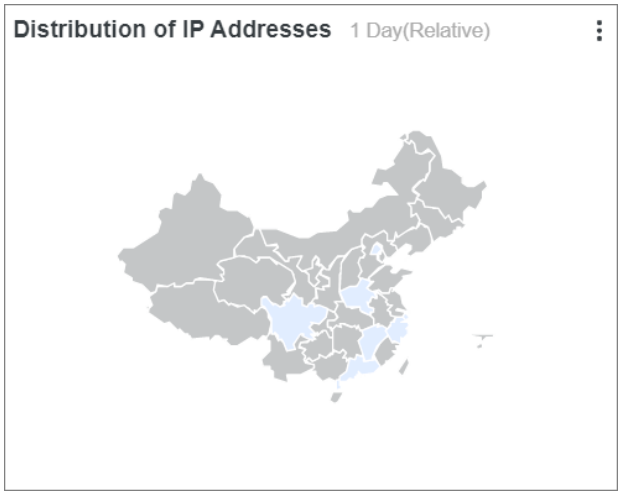
1.
2.

3. In the left-side navigation pane, choose **Log Management > Logstores**. Find the Logstore and click the **>** icon next to it.
4. Click the > icon next to **Visual Dashboards**, and then click **LogstoreName_Nginx_access_log**.

   The dashboard displays the following metrics:

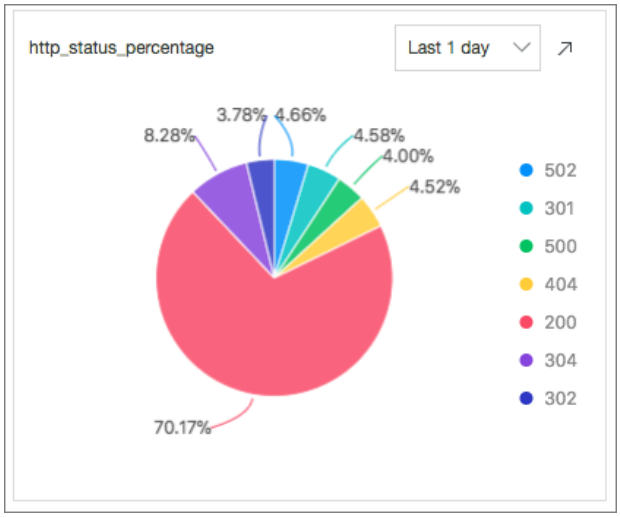
   - **Distribution of IP Addresses**: collects statistics on the distribution of IP addresses by executing the following SQL statement:

     ```
     * | select count(1) as c, ip_to_province(remote_addr) as address group by address limit 100
     ```

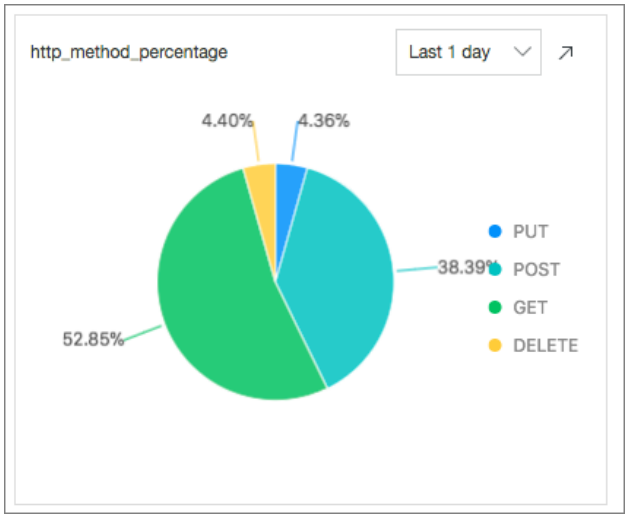
   - **HTTP Status Codes**: calculates the percentage of each HTTP status code returned in the last 24 hours by executing the following SQL statement:

     ```
     * | select count(1) as pv,
             status
             group by status
     ```

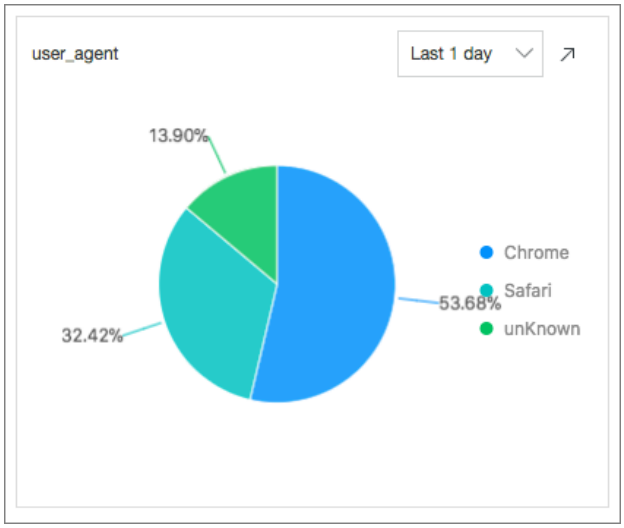
   - **Request Methods**: calculates the percentage of each request method used in the last 24 hours by executing the following SQL statement:

     ```
     * | select count(1) as pv ,request_method group by request_method
     ```



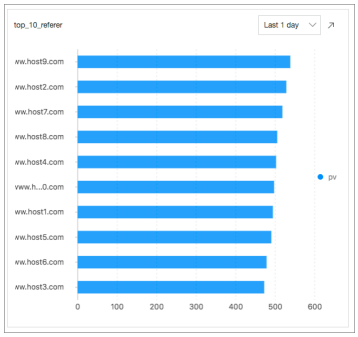
- **User Agents**: calculates the percentage of each user agent used in the last 24 hours by executing the following SQL statement:

```
* | select count(1) as pv, case when http_user_agent like '%Chrome%' then 'Chrome' when http_u
ser_agent like '%Firefox%' then 'Firefox' when http_user_agent like '%Safari%' then 'Safari' e
lse 'unKnown' end as http_user_agent  group by case when http_user_agent like '%Chrome%' then
'Chrome' when http_user_agent like '%Firefox%' then 'Firefox' when http_user_agent like '%Safa
ri%' then 'Safari' else 'unKnown' end   order by pv desc limit 10
```



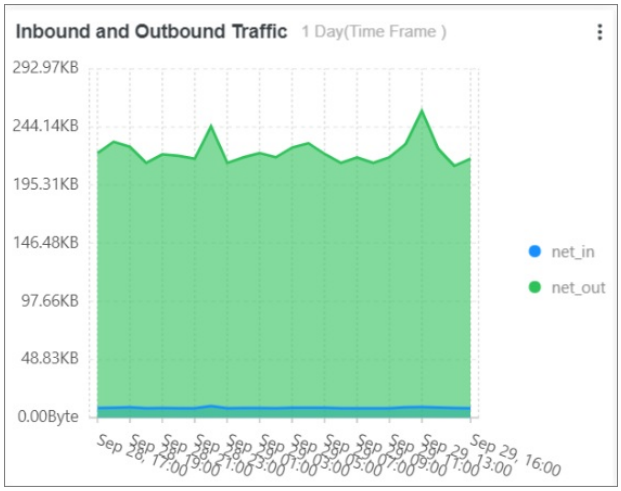
- **Top 10 Request URLs**: indicates the top 10 request URLs with the most PVs in the last 24 hours by executing the following SQL statement:

```
* | select count(1) as pv , http_referer  group by http_referer order by pv desc limit 10
```



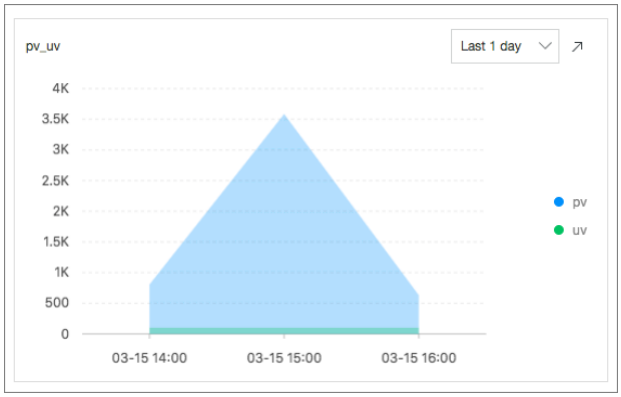
- **Inbound and Outbound Traffic**: collects statistics on the inbound and outbound traffic by executing the following SQL statement:

```
* | select sum(body_bytes_sent) as net_out, sum(request_length) as net_in ,date_format(date_trunc('hour', __time__), '%m-%d %H:%i')  as time group by date_format(date_trunc('hour', __time__), '%m-%d %H:%i') order by time limit 10000
```



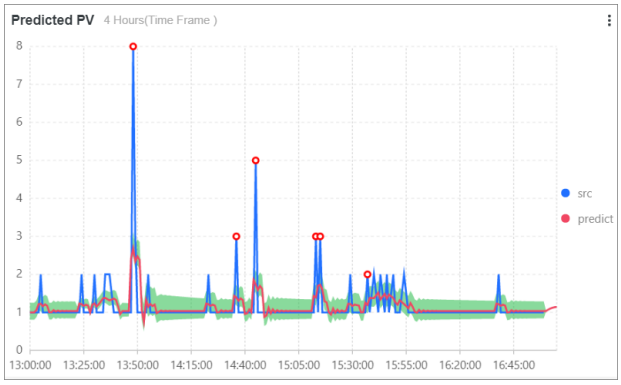
- **PV and UV Statistics**: calculates the number of PVs and UVs by executing the following SQL statement:

```
*| select approx_distinct(remote_addr) as uv ,count(1) as pv , date_format(date_trunc('hour', __time__), '%m-%d %H:%i')  as time group by date_format(date_trunc('hour', __time__), '%m-%d %H:%i')  order by time limit 1000
```



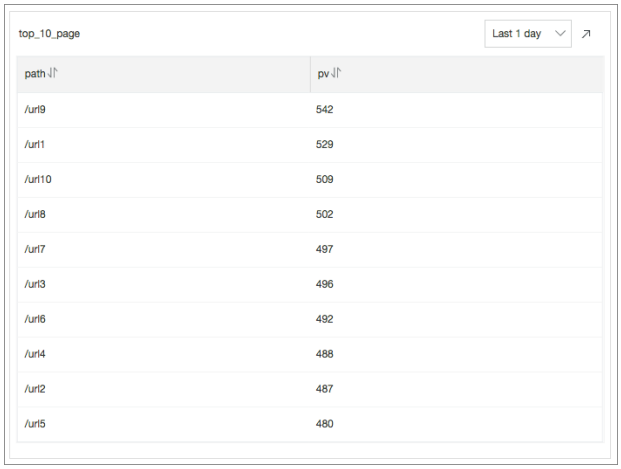
- **Predicted PV**: predicts the number of PVs in the next 4 hours by executing the following SQL statement:

```
* | select ts_predicate_simple(stamp, value, 6, 1, 'sum') from (select __time__ - __time__ % 60 as stamp, COUNT(1) as value from log GROUP BY stamp order by stamp) LIMIT 1000
```



- **Top 10 URLs by Number of Requests**: indicates the top 10 requested URLs with the most PVs in the last 24 hours by executing the following SQL statement:

```
* | select count(1) as pv, split_part(request_uri,'?',1) as path  group by path order by pv de
sc limit 10
```



Diagnose and optimize access to a website

In addition to some default access metrics, you must also diagnose access requests based on NGINX access logs. This allows you to locate requests that have high latency on specific pages. You can use the quick analysis feature on the Search & Analysis page. For more information, see Query and analyze logs.

- Count the average latency and highest latency every 5 minutes to obtain the overall latency by executing the following SQL statement:

```
* | select from_unixtime(__time__ -__time__% 300) as time,
        avg(request_time) as avg_latency ,
        max(request_time) as max_latency
        group by __time__ -__time__% 300
```

- Locate the requested page with the highest latency and optimize the response speed of the page by executing the following SQL statement:

```
* | select from_unixtime(__time__ - __time__% 60) ,
        max_by(request_uri,request_time)
        group by __time__ - __time__%60
```

- Divide all the requests into 10 groups by access latency and count the number of requests based on different latency ranges by executing the following SQL statement:

```
* |select numeric_histogram(10,request_time)
```

- Count the top 10 requests with the highest latency and the latency of each request by executing the following SQL statement:

```
* | select max(request_time,10)
```

- Optimize the requested page with the highest latency.

  Assume that the /url2 page has the highest latency. To optimize the response speed of the /url2 page, count the following metrics for the /url2 page: number of PVs and UVs, number of times that each request method is used, number of times that each HTTP status code is returned, number of times that each browser type is used, average latency, and highest latency.

```
request_uri:"/url2" | select count(1) as pv,
       approx_distinct(remote_addr) as uv,
       histogram(method) as method_pv,
       histogram(status) as status_pv,
       histogram(user_agent) as user_agent_pv,
       avg(request_time) as avg_latency,
       max(request_time) as max_latency
```

Create alert rules

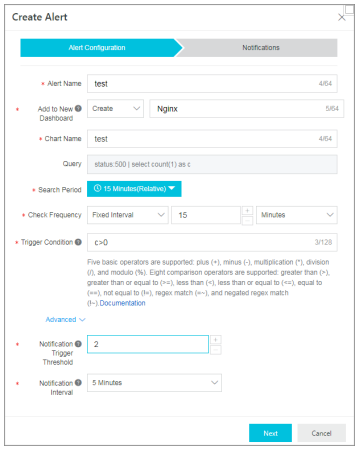
You can create alert rules to track performance issues, server errors, and traffic changes. For more information, see Configure an alert rule.

- Server alerts

  You need to focus on server errors whose HTTP status code is 500. You can execute the following SQL statement to query the error number c per unit time and set the trigger condition of the alert rule to c > 0.

  ```
  status:500 | select count(1) as c
  ```

  > **Note** For services with high access traffic, 500 errors could occur on occasion. In this case, you can set the **Notification Trigger Threshold** parameter to 2. It indicates that an alert is triggered only when conditions are met two consecutive times.



- Performance alerts

  You can create alert rules if the latency increases when the server is running. For example, you can calculate the latency of all the write requests `Post` of the operation `/adduser` by executing the following SQL statement. Then set the alert rule to l > 300000. It indicates that an alert is sent when the average latency exceeds 300 ms.

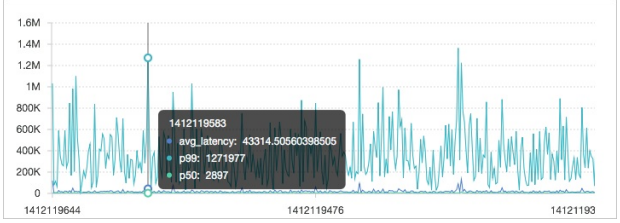
  ```
  Method:Post and URL:"/adduser" | select avg(Latency) as l
  ```

  You can use the average latency value to create an alert. However, high latency values are averaged to lower values, so this might not reflect the true situation. You can use the percentile in mathematical statistics (the highest latency is 99%) as the trigger condition.

  ```
  Method:Post and URL:"/adduser" | select approx_percentile(Latency, 0.99) as p99
  ```

  You can calculate the latency of every minute in a day (1,440 minutes), the 50th percentile latency, and the 90th percentile latency.

  ```
  * | select avg(Latency) as l, approx_percentile(Latency, 0.5) as p50, approx_percentile(Latency, 0
  .99) as p99, date_trunc('minute', time) as t group by t order by t desc limit 1440
  ```



- Traffic alerts

  A sudden decrease or increase of traffic in a short time period is abnormal. You can calculate the traffic change ratio and create alert rules to monitor sudden traffic changes. Sudden traffic changes are detected based on the following metrics:

  - Previous time period: compares data in the current time period with that in the previous time period.
  - Same time period on the previous day: compares data in the current time period with that of the same time period of the previous day.
  - Same time period of the previous week: compares data in the current time period with that of the same time period of the previous week.

  Last window is used in the following example to calculate the traffic change ratio. In this example, the time range is set to 5 minutes.

  i. Define a calculation window.

     Define a window of 1 minute to calculate the inbound traffic size of this minute.

     ```
     * | select sum(inflow)/(max(__time__)-min(__time__)) as inflow , __time__-__time__%60  as wind
     ow_time from log group by window_time order by window_time limit 15
     ```

     The result indicates that the average inbound traffic is evenly distributed in every window.

window_time	inflow
1513045740	315574947
1513045800	333233937
1513045860	335821584
1513045920	330556452
1513045980	316785257

  ii. Calculate value differences in the window.

     - Calculate the difference between the maximum or minimum traffic size and the average traffic size in the window. The max_ratio metric is used as an example.

       The calculated max_ratio is 1.02. You can set the alert rule to max_ratio > 1.5. It indicates that an alert is sent when the change ratio exceeds 50%.

       ```
        * | select max(inflow)/avg(inflow) as max_ratio from (select sum(inflow)/(max(__time__)-min
       (__time__)) as inflow , __time__-__time__%60  as window_time from log group by window_time o
       rder by window_time limit 15)
       ```

window_time	inflow
1513045740	315574947
1513045800	333233937
1513045860	335821584
1513045920	330556452
1513045980	316785257

- Calculate the latest_ratio metric to check whether the latest value fluctuates.

  Use the max_by function to calculate the maximum traffic size in the window. In this example, lastest_ratio is 0.97.

```
 * | select max_by(inflow, window_time)/1.0/avg(inflow) as lastest_ratio from (select sum(in
flow)/(max(__time__)-min(__time__)) as inflow , __time__-__time__%60  as window_time from lo
g group by window_time order by window_time limit 15)
```

  > **Note** The calculation result of the max_by function is of the character type. It must be converted to the numeric type. To calculate the relative ratio of changes, you can replace the SELECT clause with (1.0-max_by(inflow, window_time)/1.0/avg(inflow)) as lastest_ratio.

window_time	inflow
1513045740	315574947
1513045800	333233937
1513045860	335821584
1513045920	330556452
1513045980	316785257

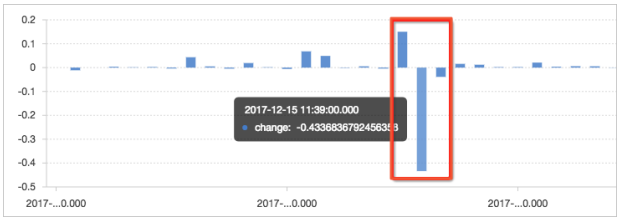
- Calculate the fluctuation ratio. It is the change ratio between the current value and the previous value of the window.

window_time	inflow
1513308660	8138522256
1513308720	8584340710
1513308780	9210706832
1513308840	9684619494

  Use the window function (lag) for calculation. Extract the current inbound traffic and the inbound traffic of the previous cycle to calculate the difference by using lag(inflow, 1, inflow)over(). Then, divide the calculated difference value by the current value to obtain the change ratio. In this example, a relatively major decrease occurs in traffic at 11:39, with a change ratio of more than 40%.

  > **Note** To define an absolute change ratio, you can use the ABS function to calculate the absolute value and unify the calculation result.

```
 * | select (inflow- lag(inflow, 1, inflow)over() )*1.0/inflow as diff, from_unixtime(window
_time) from (select sum(inflow)/(max(__time__)-min(__time__)) as inflow , __time__-__time__%
60  as window_time from log group by window_time order by window_time limit 15)
```



13.7. Analyze Apache access logs

Log Service allows you to collect Apache access logs to obtain data such as page views (PVs), unique visitors (UVs), IP address distribution, error requests, and client types. You can monitor and analyze access to your website by using Apache access logs.

Prerequisites

Apache access logs are collected. For more information, see Collect logs in Apache mode.

Indexes are created by using the data import wizard. For information about how to modify indexes, see Configure indexes.

Context

Apache is a web server software that is used to build and host websites across platforms. Apache access logs can be collected and analyzed.

In the Log Service console, you can create a collection configuration to collect Apache access logs by using the data import wizard. Then, Log Service creates indexes and an Apache dashboard to collect and analyze Apache access logs. The dashboard displays metrics such as Distribution of IP Addresses, HTTP Status Codes, Request Methods, PV and UV Statistics, Inbound and Outbound Traffic, User Agents, Top 10 Request URLs, Top 10 URIs by Number of Requests, and Top 10 URIs by Request Latency.

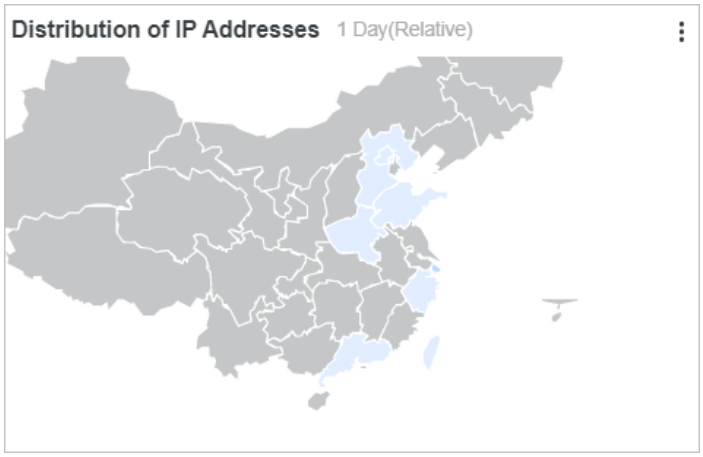
Procedure

1.
2.
3. In the left-side navigation pane, choose **Log Management > Logstores** . Find the Logstore and click the **>** icon next to it.
4. Click the > icon next to **Visual Dashboards**, and then click **LogstoreName_apache_access_log**.

   The dashboard displays the following metrics:

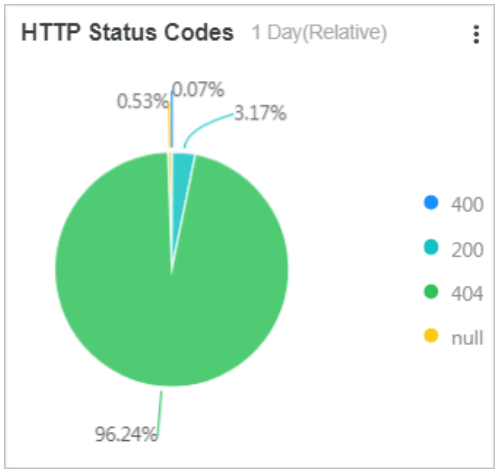
   - **Distribution of IP Addresses**: indicates the distribution of IP addresses by executing the following SQL statement:

     ```
     * | select ip_to_province(remote_addr) as address, count(1) as c group by ip_to_province(remote_addr) limit 100
     ```

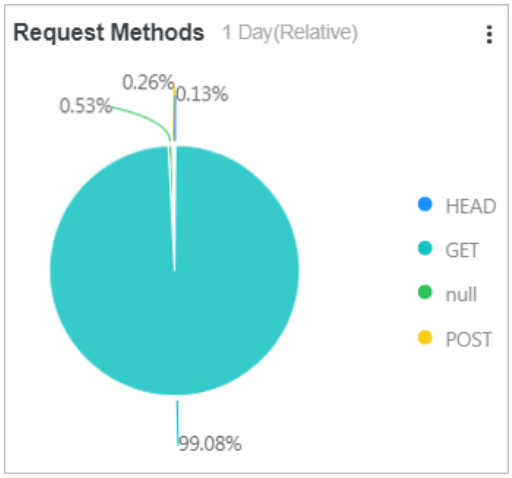
   - **HTTP Status Codes**: indicates the percentage of each HTTP status code returned within the last day by executing the following SQL statement:

     ```
     * | select status, count(1) as pv group by status
     ```



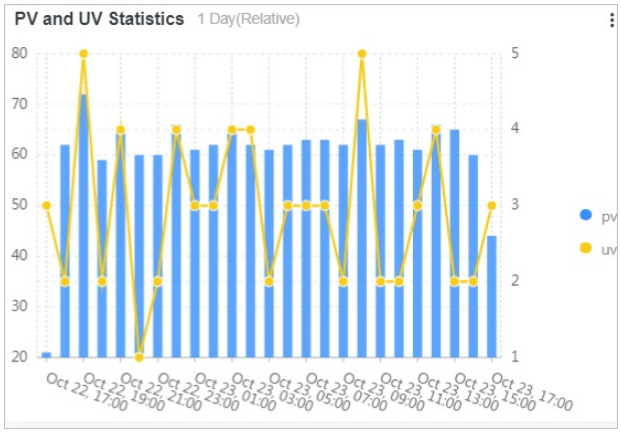
- **Request Methods**: indicates the percentage of each request method used within the last day by executing the following SQL statement:

```
* | select request_method, count(1) as pv group by request_method
```



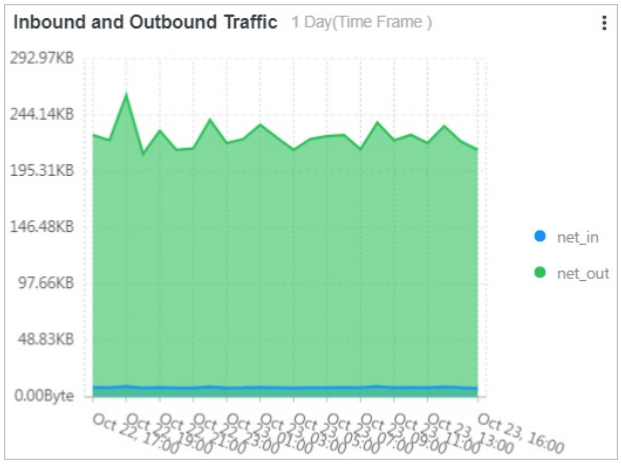
- **PV and UV Statistics**: indicates the number of PVs and UVs by executing the following SQL statement:

```
* | select date_format(date_trunc('hour', __time__), '%m-%d %H:%i')  as time, count(1) as pv,
approx_distinct(remote_addr) as uv group by date_format(date_trunc('hour', __time__), '%m-%d %
H:%i') order by time limit 1000
```



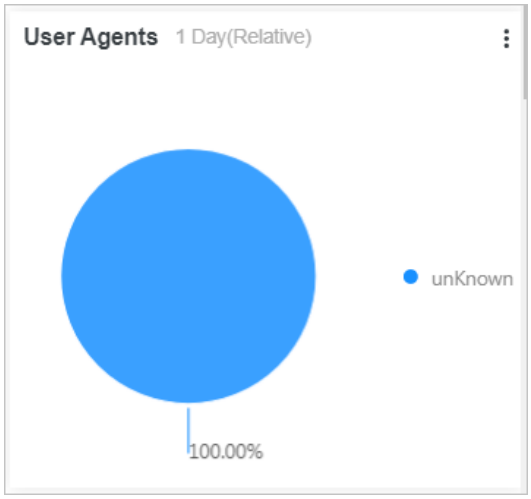
- **Inbound and Outbound Traffic**: indicates the inbound and outbound traffic by executing the following SQL statement:

```
* | select date_format(date_trunc('hour', __time__), '%m-%d %H:%i') as time, sum(bytes_sent) a
s net_out, sum(bytes_received) as net_in group by time order by time limit 10000
```



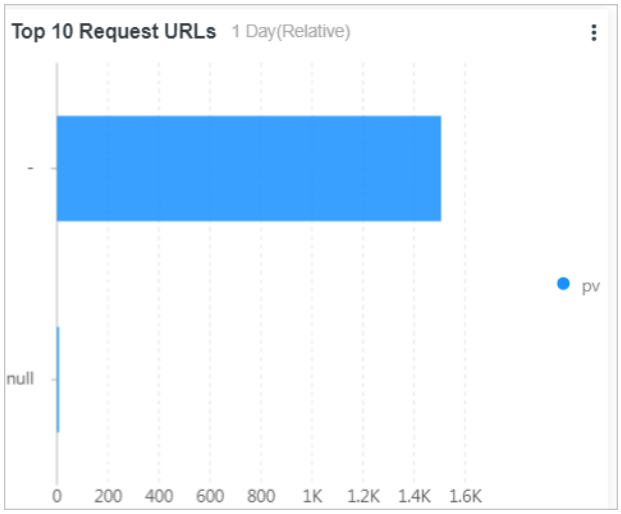
- **User Agents**: indicates the percentage of each user agent used within the last day by executing the following SQL statement:

```
* | select case when http_user_agent like '%Chrome%' then 'Chrome' when http_user_agent like '
%Firefox%' then 'Firefox' when http_user_agent like '%Safari%' then 'Safari' else 'unKnown' en
d as http_user_agent, count(1) as pv group by case when http_user_agent like '%Chrome%' then '
Chrome' when http_user_agent like '%Firefox%' then 'Firefox' when http_user_agent like '%Safar
i%' then 'Safari' else 'unKnown' end   order by pv desc limit 10
```



- **Top 10 Request URLs**: indicates the top 10 request URLs that have the most PVs within the last day by executing the following SQL statement:

```
* | select  http_referer, count(1) as pv group by http_referer order by pv desc limit 10
```



- **Top 10 URLs by Number of Requests**: indicates the top 10 requested URLs that have the most PVs within the last day by executing the following SQL statement:

```
* | select split_part(request_uri,'?',1) as path,  count(1) as pv group by split_part(request_
uri,'?',1) order by pv desc limit 10
```

Top 10 URIs by Number of Requests 1 Day(Relative)

pv	path
1439	/private/nginx_status
46	/
8	null
2	/GponForm/diag_Form
2	/img/centos-logo.png
2	/boaform/admin/formLogin
2	/favicon.ico
1	/portal/redlion
1	/streaming/clients_live.php
1	/.env

- **Top 10 URLs by Request Latency**: indicates the top 10 requested URLs with the highest latency within the last day by executing the following SQL statement:

```
* | select request_uri as top_latency_request_uri,
            request_time_sec
            order by request_time_sec desc limit 10 10
```

Top 10 URIs by Request Latency 1 Day(Relative)

top_latency_request_uri	request_time
/	0.412
/sdk	0.1
/	0.08
/	0.076
/	0.076
/	0.072
/	0.071
/	0.068
/	0.068
/	0.063

13.8. Analyze IIS access logs

Log service allows you to collect and analyze Internet Information Services (IIS) access logs. This topic describes how to monitor and analyze access to your website by using IIS access logs. You can obtain data such as page views (PVs), unique visitors (UVs), client IP distribution, error requests, and inbound and outbound traffic.

Prerequisites

IIS logs are collected. For more information, see Collect logs in IIS mode.

The indexing feature is enabled and configured. For more information, see Configure indexes.

Context

IIS is a secure web server that you can use to build and host websites. When you use IIS to build a website, you can collect and analyze IIS access logs.

We recommend that you use the following IIS W3C Extended Log Format:

```
logExtFileFlags="Date, Time, ClientIP, UserName, SiteName, ComputerName, ServerIP, Method, UriStem, U
riQuery, HttpStatus, Win32Status, BytesSent, BytesRecv, TimeTaken, ServerPort, UserAgent, Cookie, Ref
erer, ProtocolVersion, Host, HttpSubStatus"
```

The following example shows how to obtain IIS logs in the IIS W3C Extended Log Format:

```
#Software: Microsoft Internet Information Services 7.5
#Version: 1.0
#Date: 2020-09-08 09:30:26
#Fields: date time s-sitename s-ip cs-method cs-uri-stem cs-uri-query s-port cs-username c-ip cs(User
-Agent) sc-status sc-substatus sc-win32-status sc-bytes cs-bytes time-taken
2009-11-26 06:14:21 W3SVC692644773 125.67.67. * GET /index.html - 80 - 192.0.2.0 Baiduspider+(+http:/
/www.example.com)200 0 64 185173 296 0
```

- Field prefixes

Prefix	Description
s-	Indicates a server action.
c-	Indicates a client action.
cs-	Indicates a client-to-server action.
sc-	Indicates a server-to-client action.

- Fields

Field	Description
date	The date on which the client sends the request.
time	The time when the client sends the request.
s-sitename	The Internet service name and instance number of the site visited by the client.
s-computername	The name of the server on which the log entry is generated.
s-ip	The IP address of the server on which the log entry is generated.
cs-method	The HTTP request method used by the client, for example, GET or POST.

Field	Description
cs-uri-stem	The URI resource requested by the client.
cs-uri-query	The query string that follows the question mark (?) in the HTTP request.
s-port	The port number of the server to which the client is connected.
cs-username	The username that the client uses to access the server. Authenticated users are referenced as `domain\username`. Anonymous users are indicated by a hyphen (-).
c-ip	The IP address of the client that sends the request.
cs-version	The protocol version used by the client, for example, HTTP 1.0 or HTTP 1.1.
cs(User-Agent)	The browser used by the client.
Cookie	The content of the sent cookie or received cookie. A hyphen (-) is used if no cookie is sent or received.
referer	The site that the client last visited.
cs-host	The header name of the host.
sc-status	The HTTP or FTP status code returned by the server.
sc-substatus	The HTTP sub-status code returned by the server.
sc-win32-status	The Windows status code returned by the server.
sc-bytes	The number of bytes sent by the server.
cs-bytes	The number of bytes received by the server.
time-taken	The processing time of the request. Unit: milliseconds.

Procedure

1.
2.
3.
4. In the upper-right corner of the page, click **15 Minutes (Relative)** to set a time range for the query.

   You can select a relative time, set a time frame, or customize a time range.

   > **Note** The query results may contain logs that are generated one minute earlier or later than the specified time range.

5. Enter a query statement in the search box, and then click **Search & Analyze**.

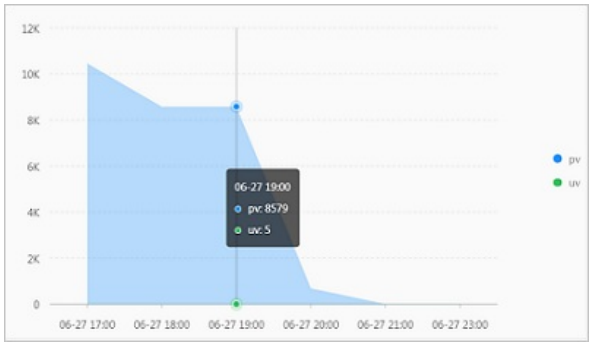
   For more information, see 查询概述.

   - Collect statistics on the distribution of client IP addresses by executing the following SQL statement:

     ```
     | select ip_to_geo("c-ip") as country, count(1) as c group by ip_to_geo("c-ip") limit 100
     ```

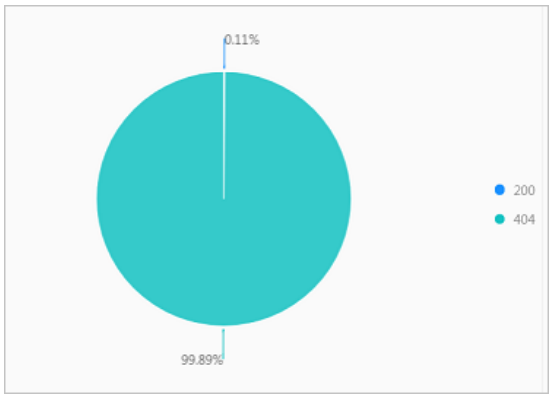
   - Calculate the number of PVs and UVs by executing the following SQL statement:

```
*| select approx_distinct("c-ip") as uv ,count(1) as pv , date_format(date_trunc('hour', __time__), '%m-%d %H:%i') as time group by date_format(date_trunc('hour', __time__), '%m-%d %H:%i') order by time limit 1000
```



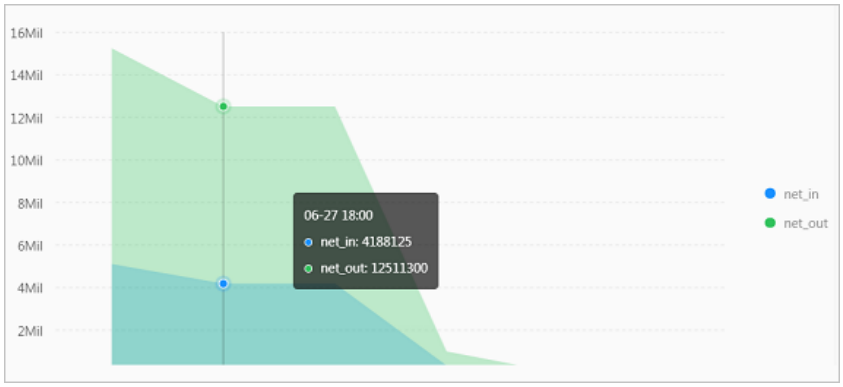
- Calculate the percentage of each HTTP status code returned by executing the following SQL statement:

```
*| select count(1) as pv ,"sc-status" group by "sc-status"
```



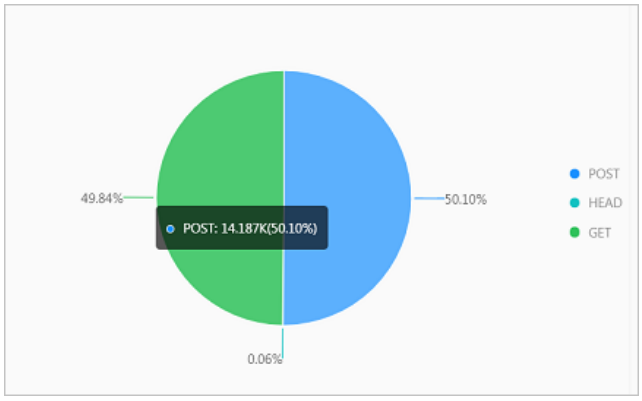
- Collect statistics on the inbound and outbound traffic by executing the following SQL statement:

```
*| select sum("sc-bytes") as net_out, sum("cs-bytes") as net_in ,date_format(date_trunc('hour', time), '%m-%d %H:%i') as time group by date_format(date_trunc('hour', time), '%m-%d %H:%i') order by time limit 10000
```



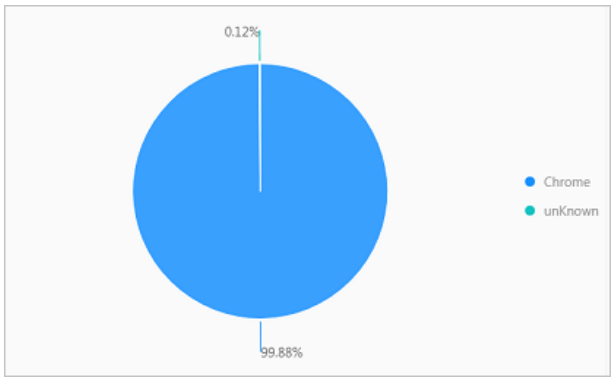
- Calculate the percentage of each request method by executing the following SQL statement:

```
*| select count(1) as pv ,"cs-method" group by "cs-method"
```



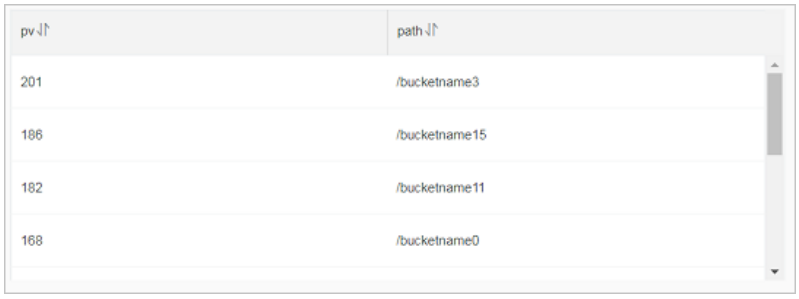
- Calculate the percentage of each browser type by executing the following SQL statement:

```
*| select count(1) as pv, case when "user-agent" like '%Chrome%' then 'Chrome' when "user-agen
t" like '%Firefox%' then 'Firefox' when "user-agent" like '%Safari%' then 'Safari' else 'unKno
wn' end as "user-agent" group by case when "user-agent" like '%Chrome%' then 'Chrome' when "us
er-agent" like '%Firefox%' then 'Firefox' when "user-agent" like '%Safari%' then 'Safari' else
'unKnown' end order by pv desc limit 10
```



- Calculate the top 10 pages that have the most PVs by executing the following SQL statement:

```
*| select count(1) as pv, split_part("cs-uri-stem",'?',1) as path group by split_part("cs-uri-
stem",'?',1) order by pv desc limit 10
```



pv	path
201	/bucketname3
186	/bucketname15
182	/bucketname11
168	/bucketname0

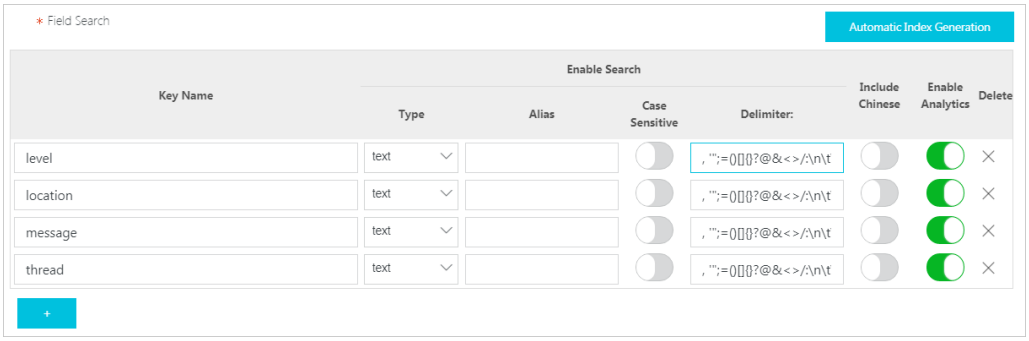
13.9. Analyze Log4j logs

This topic describes how to analyze Log4j logs in the Log Service console. The logs of an e-commerce company are used as an example.

Prerequisites

- Log4j logs are collected. For more information, see Collect Log4j logs.
- The indexing feature is enabled for the Logstore and indexes are configured. For more information, see Configure indexes.

The following figure shows the indexes that are used in this example.



Context

Log4j is an open source project of Apache. Log4j allows you to specify the output destination and format of logs. You can also specify the severity level of logs. The severity levels of logs are classified into ERROR, WARN, INFO, and DEBUG in descending order. The output destination specifies whether logs are sent to the console or files. The output format specifies the format of logs.

In this example, an e-commerce company wants to obtain the best solution for the platform. The company needs to analyze information that includes behavioral data, such as logon methods, logon time, logon duration, accessed pages, access duration, average order time, and consumption level, platform stability, system errors, and data security. Log Service provides multiple log collection methods and the log analysis feature. Sample logs are collected by Log Service, as shown in the following examples.

- The following log indicates the logon information:

```
level:  INFO
location:  com.aliyun.log4jappendertest.Log4jAppenderBizDemo.login(Log4jAppenderBizDemo.java:38)
message:  User login successfully. requestID=id4 userID=user8
thread:  main
time:  2018-01-26T15:31+0000
```

- The following log indicates the purchase information:

```
level:  INFO
location:  com.aliyun.log4jappendertest.Log4jAppenderBizDemo.order(Log4jAppenderBizDemo.java:46)
message:  Place an order successfully. requestID=id44 userID=user8 itemID=item3 amount=9
thread:  main
time:  2018-01-26T15:31+0000
```

Procedure

1.
2.
3.
4. In the upper-right corner of the page, click **15 Minutes (Relative)** to specify a time range for the query.

   You can select a relative time or time frame. You can also specify a custom time range.

   > **Note** The query results may contain logs that are generated 1 minute earlier or later than the specified time range.

5. Enter a query statement in the search box, and then click **Search & Analyze**.

   For more information about query statements, see 查询概述.

   - You can enter the following query statement to view the statistics of the three locations where the most errors occurred in the last hour:

```
level: ERROR | select location ,count(*) as count GROUP BY  location  ORDER BY count DESC LIMI
T 3
```

- You can enter the following query statement to view the number of log entries that are generated in each severity level in the last 15 minutes:

```
| select level ,count(*) as count GROUP BY level ORDER BY count DESC
```

- You can enter the following query statement to view the statistics of the three user IDs that log on to the platform for the most number of times in the last hour:

```
login | SELECT regexp_extract(message, 'userID=(? <userID>[a-zA-Z\d]+)', 1) AS userID, count(*
) as count GROUP BY userID ORDER BY count DESC LIMIT 3
```

- You can enter the following query statement to view the total payment amount of each user ID in the last 15 minutes:

```
order | SELECT regexp_extract(message, 'userID=(? <userID>[a-zA-Z\d]+)', 1) AS userID, sum(cas
t(regexp_extract(message, 'amount=(? <amount>[a-zA-Z\d]+)', 1) AS double)) AS amount GROUP BY
userID
```

13.10. Query and analyze application logs

This topic describes how to use Log Service to query and analyze application logs in different scenarios, such as log query, association analysis, and statistical analysis.

Context

Application logs include important statistical information about application operation and maintenance. Application logs have the following features:

- Inconsistent log style. Application developers have different styles when they develop code. The logs that are generated by different applications have inconsistent styles.
- Large data size. The size of application logs is one order of magnitude larger than that of access logs.
- Multiple distributed servers. Most applications are stateless and run on different frameworks, such as Elastic Compute Service (ECS) and Container Service. These applications may be deployed on a few to thousands of instances. Therefore, a cross-server solution for log collection is required.
- Complex runtime environments. Applications are executed in different environments and relevant logs are stored in different environments. For example, application-related logs are stored in containers, API-related logs are stored in Function Compute, old system logs are stored in data centers, mobile app logs are stored in mobile terminals, and website logs are stored in browsers.

To obtain full logs, all application logs must be stored in the same environment. Log Service provides multiple log collection methods and the log analysis feature. You can analyze logs in real time by using the query statements and SQL-92 syntax. You can also visualize the query results on charts. The cost of the solution that is provided by Log Service is only 25% of the cost of open source solutions.

Query application logs

In this example, an order error or request latency occurs when an application is used. You can use search statements to locate the issue in logs that contain terabytes of data within 1 second. To obtain a precise query result, you can specify a time range and keywords based on your business requirements.

- You can run the following search statement to query the log entries of requests whose latency is more than 1 second and whose request method starts with Post:

```
Latency > 1000000 and Method=Post*
```

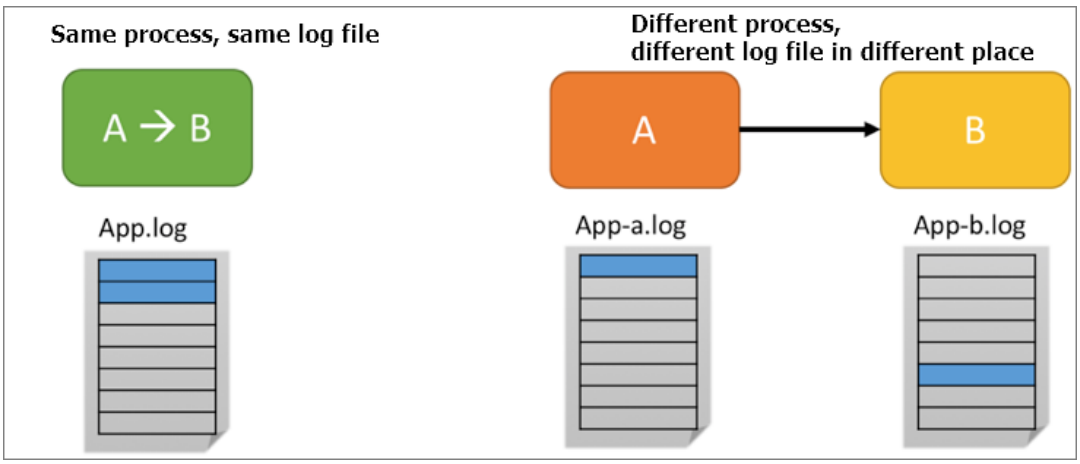
- You can run the following search statement to query the log entries whose keywords include error and exclude merge:

```
error not merge
```

Perform an association analysis on application logs

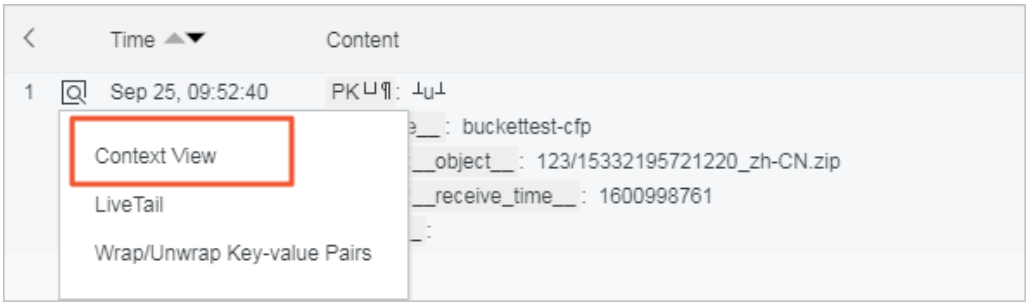
The types of association analysis include the intra-process association analysis and the cross-process association analysis. The two types of association analysis have the following differences:

- Intra-process association analysis: The logs of a process are stored in the same log file. In a multi-threaded process, you can filter logs based on thread IDs.
- Cross-process association analysis: The association between multiple processes are unclear. The processes are associated based on the TraceId parameter. The value of the TraceId parameter is automatically generated when you use a remote procedure call (RPC) to send a request.

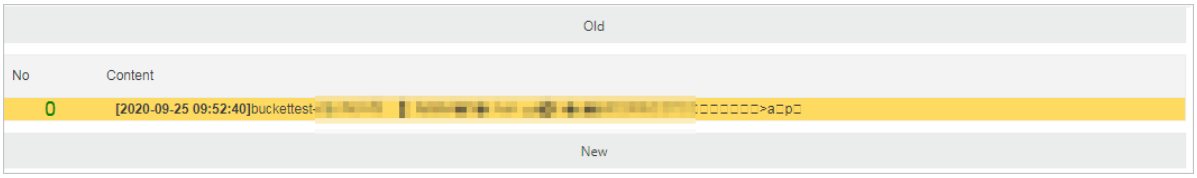


- Intra-process association analysis

  View the associated logs by using the context query feature. You can query an exception log entry by entering a keyword, and then click **Context View** to view the log entries that are obtained before and after the exception log entry. For more information, see Context query.



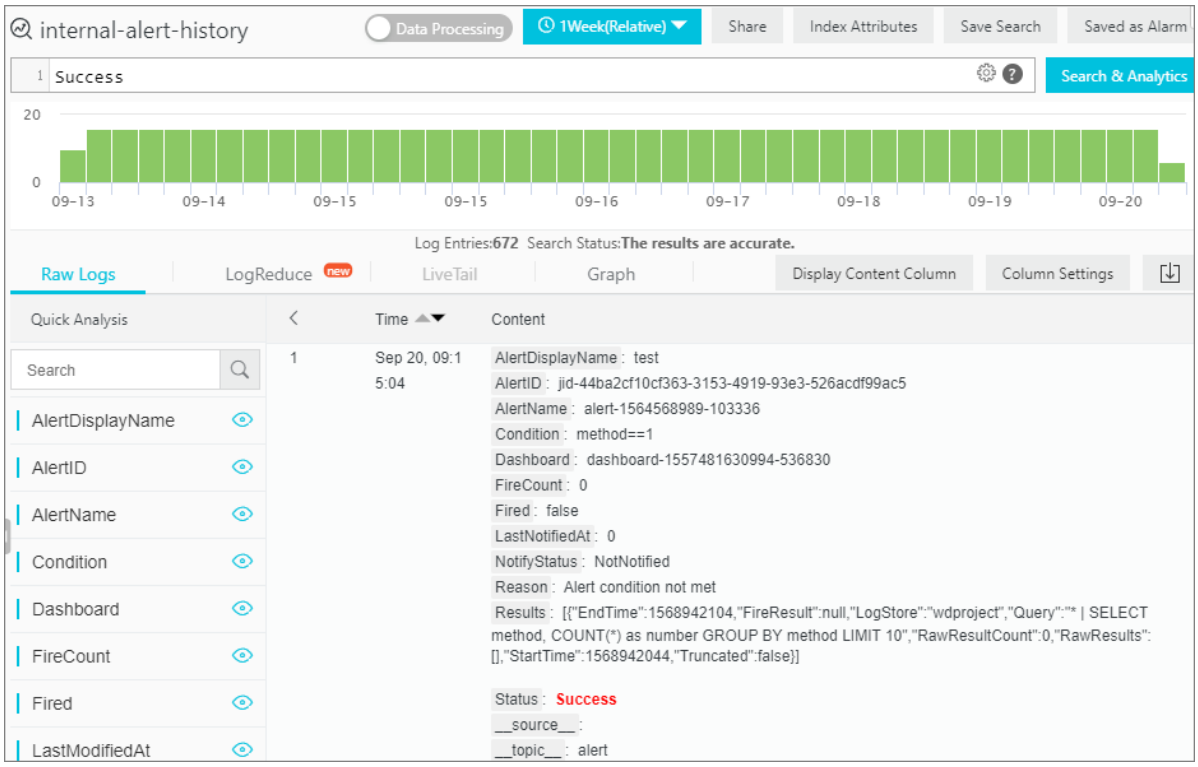
  The following figure shows the context query results.



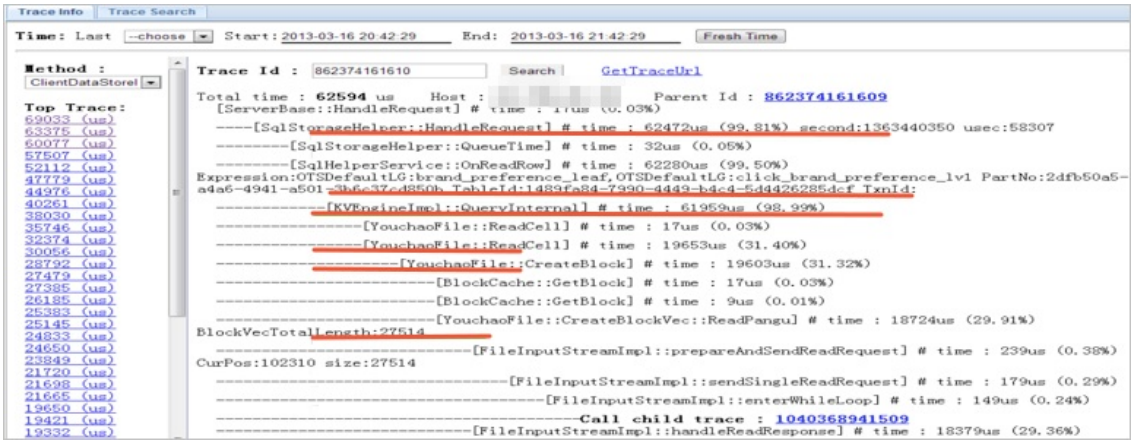
- Cross-process association analysis

  The cross-process association analysis feature has the same feature as tracing tools such as EagleEye, Dapper, StackDriver Trace, Zipkin, Appdash, and X-ray.

The cross-process analysis feature implements the basic tracing feature based on Log Service. To obtain logs from different Logstores, you can configure log fields that can be associated when you collect logs from different modules, for example, the request_id field and the order_id field.



You can use SDKs to query logs that are collected from different modules, such as frontend servers, backend servers, payment systems, and ordering systems. After you obtain the query results, you can create a frontend page to associate the results, as shown in the following figure.

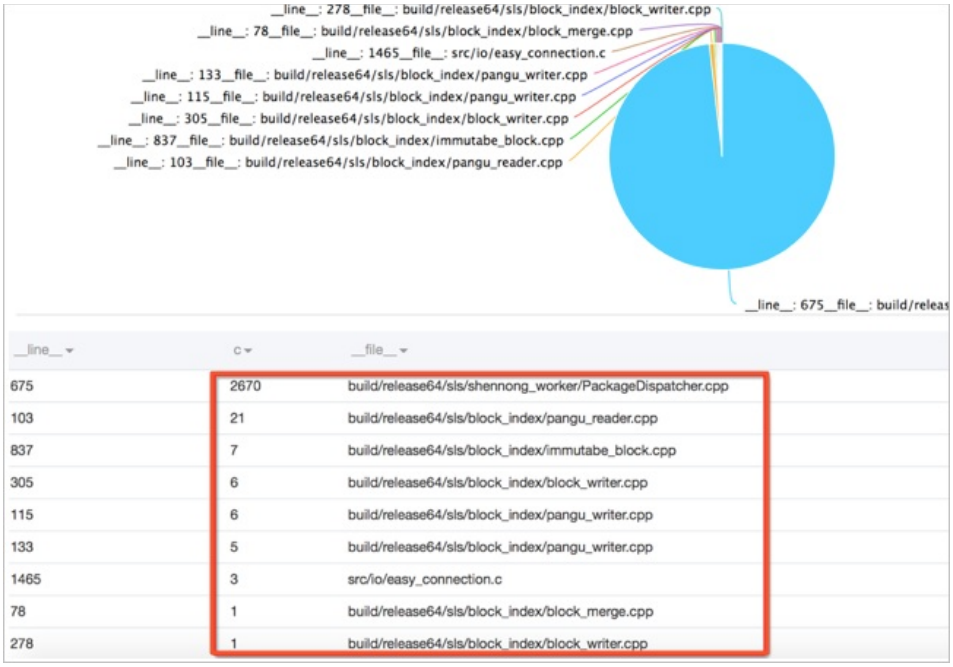


Perform a statistical analysis on application logs

After you obtain the query results, you can also perform a statistical analysis on the obtained logs.

You can run the following query statement to view the statistics of all error types and the distribution of all error locations:

```
__level__:error | select __file__, __line__, count(*) as c group by __file__, __line__ order by c des
c
```



__line__	c	__file__
675	2670	build/release64/sls/shennong_worker/PackageDispatcher.cpp
103	21	build/release64/sls/block_index/pangu_reader.cpp
837	7	build/release64/sls/block_index/immutabe_block.cpp
305	6	build/release64/sls/block_index/block_writer.cpp
115	6	build/release64/sls/block_index/pangu_writer.cpp
133	5	build/release64/sls/block_index/pangu_writer.cpp
1465	3	src/io/easy_connection.c
78	1	build/release64/sls/block_index/block_merge.cpp
278	1	build/release64/sls/block_index/block_writer.cpp

What to do next

- Back up logs.

  You can back up the obtained logs to other cloud services such as Object Storage Service (OSS) and MaxCompute.

- Configure alerts.

  You can also use Cloud Monitor to configure alerts for the obtained logs.

- Grant permissions.

  You can grant permissions to a RAM user or user group to isolate the development environment and the production environment.

13.11. Analyze website logs

You can use the SQL-92 syntax to analyze logs in Log Service. You can also visualize all query results on multiple types of charts, such as table, line chart, column chart, pie chart, flow chart, and map. This topic describes how to analyze website logs in the Log Service console and visualize the query results on charts.

Prerequisites

- Website logs are collected. For more information, see Log collection methods.
- The indexing feature is enabled for the Logstore and indexes are configured. For more information, see Configure indexes.

Context

Website logs include important statistical information about website operation and maintenance, such as page views (PVs), unique visitors (UVs), distribution of accessed regions, and top 10 accessed websites. Log Service provides multiple log collection methods and the log analysis feature. You can analyze logs in real time by using the query statements and SQL-92 syntax. You can also visualize the query results on charts. Log Service also allows you to visualize log analysis results on built-in dashboards or by using open-source visualization tools such as DataV, Grafana, Tableau through Java Database Connectivity (JDBC), and Quick BI.

Procedure

1.
2.
3.
4. In the upper-right corner of the page, click **15 Minutes (Relative)** to specify a time range for the query.

   You can select a relative time or time frame. You can also specify a custom time range.

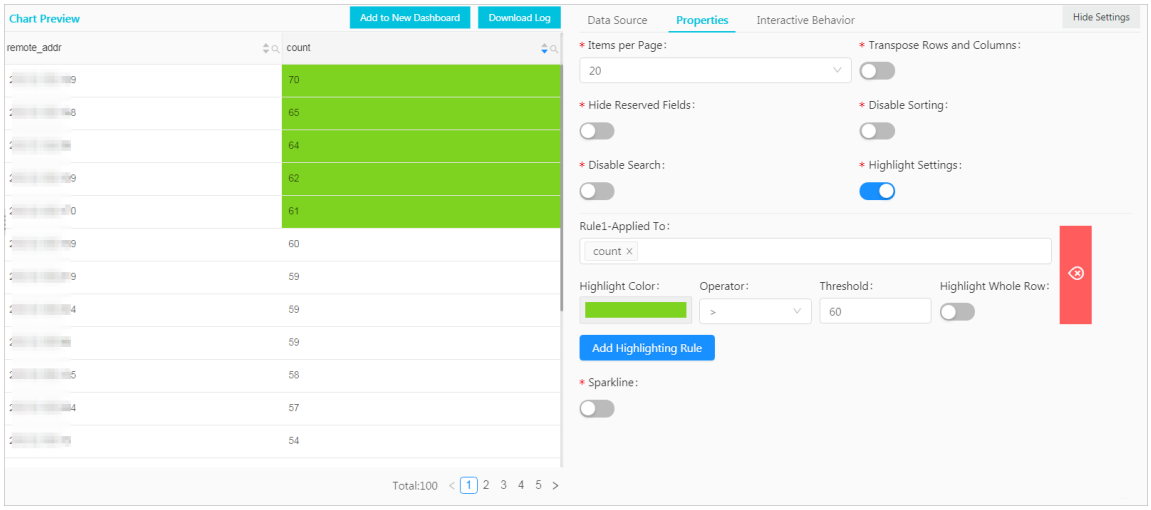
   > **Note** The query results may contain logs that are generated 1 minute earlier or later than the specified time range.

5. Enter a query statement in the search box, and then click **Search & Analyze**.

   Log Service provides multiple charts to display query results. For more information, see Charts.

   - Table. You can enter the following query statement to display the access statistics of the client IP addresses in the last day and sort the query results in descending order. The remote_addr field indicates the IP address of a client that sends an access request.

     ```
     * | SELECT remote_addr, count(*) as count GROUP BY remote_addr ORDER BY count DESC
     ```

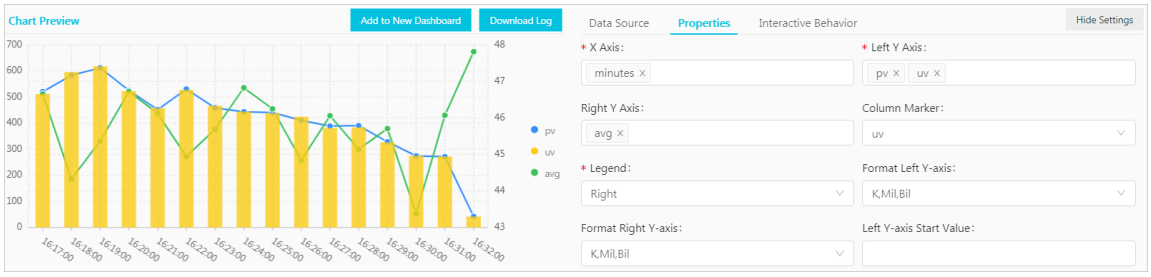
     

   - Line chart. You can enter the following query statement to display the statistics of PVs, UVs, and average

response time in the last 15 minutes:

```
* | select date_format(from_unixtime(__time__ - __time__ % 60), '%H:%i:%S') as minutes, approx_distinct(remote_addr) as uv, count(1) as pv, avg(request_time) as avg group by minutes order by minutes asc limit 100000
```

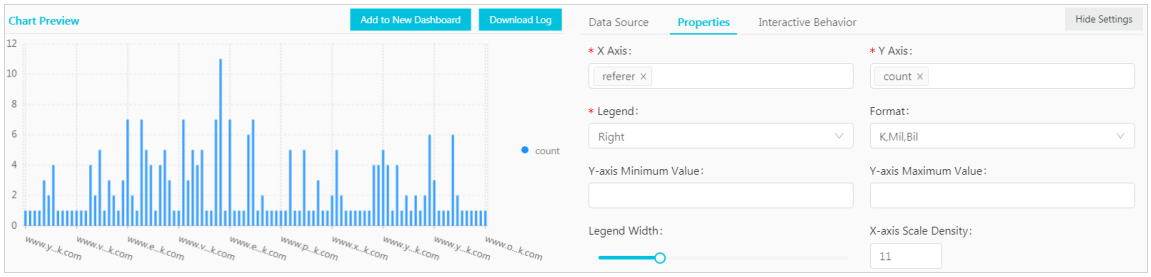
Select *minutes* for **X Axis**, select *pv* and *uv* for **Left Y Axis**, select *avg* for **Right Y Axis**, and select *uv* for **Column Marker**. The following figure shows the line chart and properties.



- Column chart. You can enter the following query statement to display the number of the source URLs that are accessed in the last 15 minutes. The referer field indicates the HTTP referer header. This field includes the source URL information.

```
* | select referer, count(1) as count group by referer
```

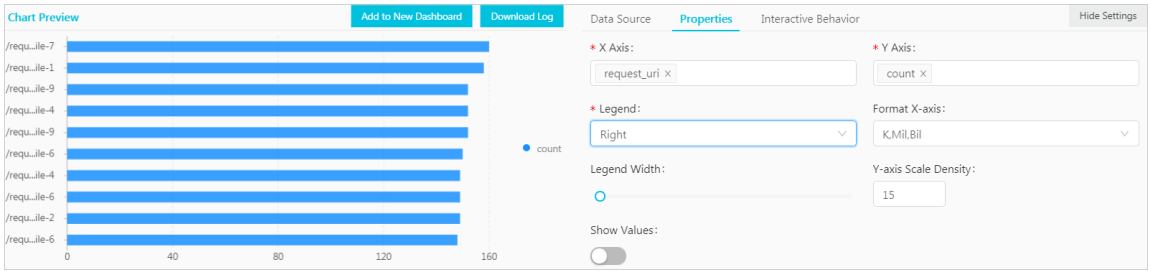
Select *referer* for **X Axis** and select *count* for **Y Axis**. The following figure shows the column chart and properties.



- Bar chart. You can enter the following query statement to display the statistics of the top 10 accessed websites in the last 15 minutes. The request_uri field indicates the URI of a request.

```
* | select  request_uri, count(1) as count group by request_uri order by count desc limit 10
```

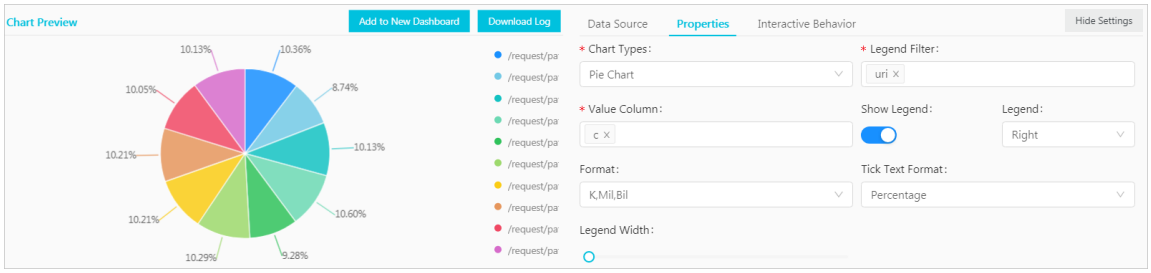
Select *request_uri* for **X Axis** and select *count* for **Y Axis**. The following figure shows the bar chart and properties.



- Pie chart. You can enter the following query statement to display the statistics of the accessed websites in the last 15 minutes. The request_uri field indicates the URI of a request.

```
* | select request_uri as uri , count(1) as c group by uri limit 10
```

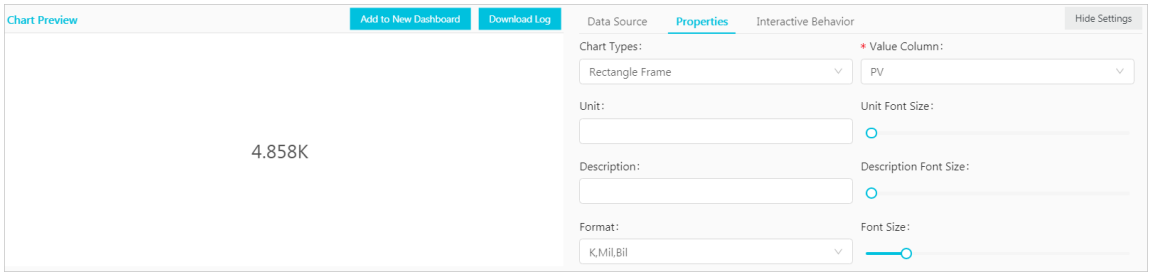
Select *uri* for **Legend Filter** and select *c* for **Value Column**. The following figure shows the pie chart and properties.



- Single value chart. You can enter the following query statement to display the number of PVs in the last 15 minutes:

```
* | select count(1) as PV
```

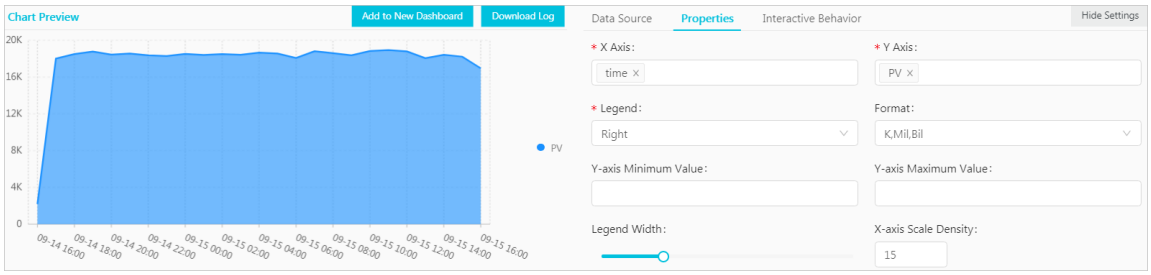
Select *PV* for **Value Column**. The following figure shows the single value chart and properties.



- Area chart. You can enter the following query statement to display the access statistics of an IP address in the last day:

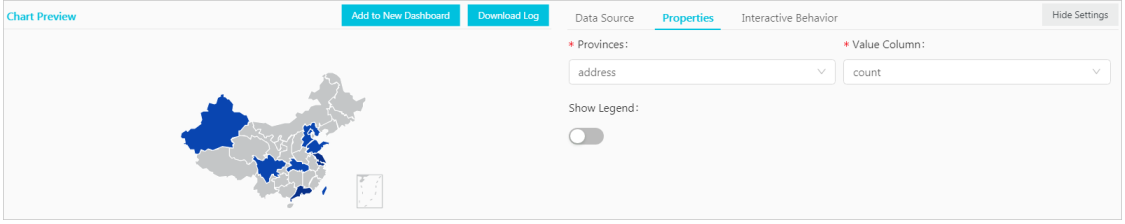
```
remote_addr: 10.0.XX.XX | select date_format(date_trunc('hour', __time__), '%m-%d %H:%i') as t
ime, count(1) as PV group by time order by time limit 1000
```

Select *time* for **X Axis** and select *PV* for **Y Axis**. The following figure shows the area chart and properties.



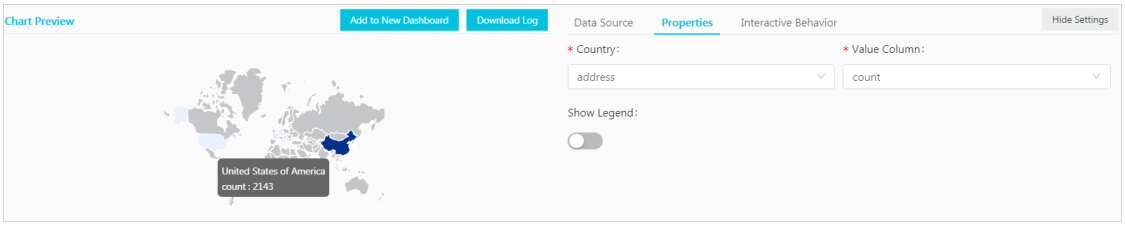
- Map. You can enter the following query statements to display the top 10 locations to which the client IP addresses belong:
  - Map of China

```
* | select ip_to_province(remote_addr) as address, count(1) as count group by address order
by count desc limit 10
```



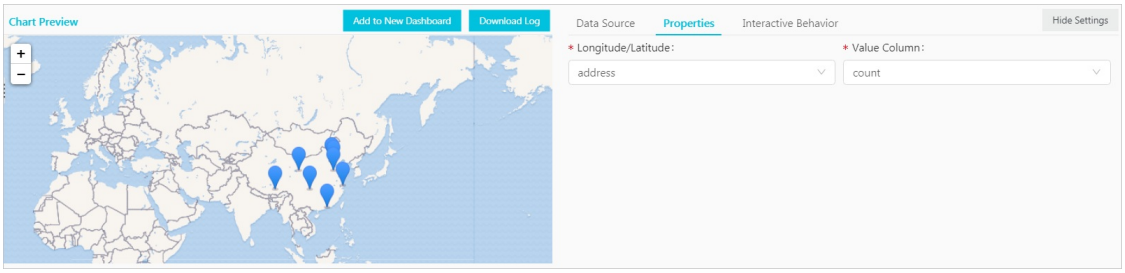
- World Map

```
* | select ip_to_country(remote_addr) as address, count(1) as count group by address order b
y count desc limit 10
```



- AMap

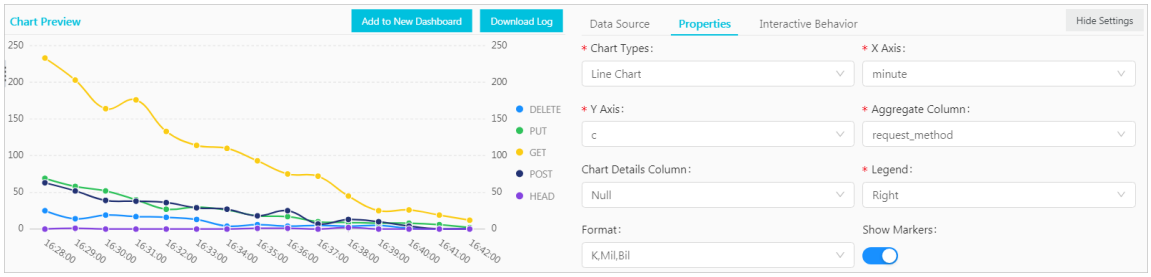
```
* | select ip_to_geo(remote_addr) as address, count(1) as count group by address order by co
unt desc limit 10
```



- Flow chart. You can enter the following query statement to display the times trends of the request methods that are used in the last 15 minutes. The request_method field indicates the method of an HTTP access request.

```
* | select date_format(from_unixtime(__time__ - __time__% 60), '%H:%i:%S') as minute, count(1)
as c, request_method group by minute, request_method order by minute asc limit 100000
```

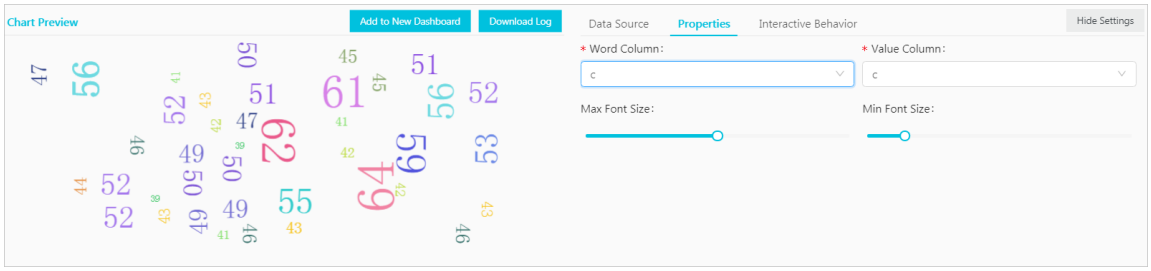
Select *minute* for **X Axis**, select *c* for **Y Axis**, and select *request_method* for **Aggregate Column**. The following figure shows the flow chart and properties.



- Word cloud. You can enter the following query statement to display the statistics of the accessed websites in the last 15 minutes. The request_uri field indicates the URI of a request.

```
* | select request_uri as uri , count(1) as c group by uri limit 100
```

Select *c* for **Word Column** and select *c* for **Value Column**. The following figure shows the word cloud and properties.



6. Add an analysis chart to a dashboard.

   You can click **Add to New Dashboard** to perform this operation. For more information, see Add charts to a dashboard.

13.12. Analyze layer-7 access logs of SLB

This topic describes how to use the query and analysis feature of Log Service to analyze layer-7 access logs of Server Load Balancer (SLB) in real time.

Prerequisites

Layer-7 access logs of SLB are collected. For more information, see Enable the access log management feature.

Context

SLB is a basic component for most cloud services. In most cases, you need to continuously monitor, detect, diagnose, and configure alerts for SLB. Alibaba Cloud SLB distributes traffic to multiple ECS instances to improve the service capabilities of applications. You can use SLB to prevent single point of failures (SPOFs) and a large number of concurrent web access requests.

Access logs can be generated for layer-7 SLB based on HTTP or HTTPS. For more information about access logs, see Log fields. SLB has the following metrics:

- Page Views (PVs): the total number of HTTP or HTTPS requests sent by the clients.
- Unique Visitors (UVs): the total number of unique requests. Requests initiated by unique visitors from the same client are counted only once.
- Request success rate: the percentage of the requests whose status code is 2XX to the total PVs.
- Request traffic: the total number of bytes of request messages that are sent by the clients.
- Response traffic: the total number of bytes of the HTTP message body that is sent to the clients.
- PV heat map: indicates the density of PVs in the regions where the IP addresses of the clients reside.

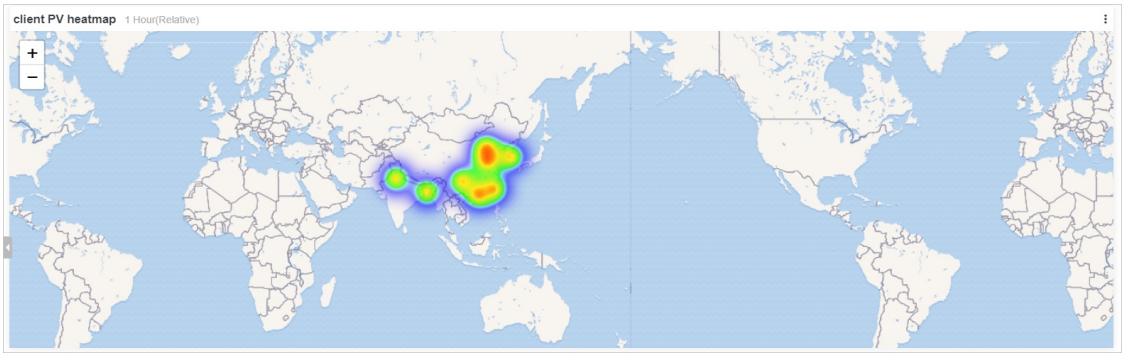
Examples

1.
2.
3. In the left-side navigation pane, choose **Log Management > Logstores**. Find the destination Logstore and click the **>** icon next to it.
4. In the **Visual Dashboards** section, click the destination dashboard.

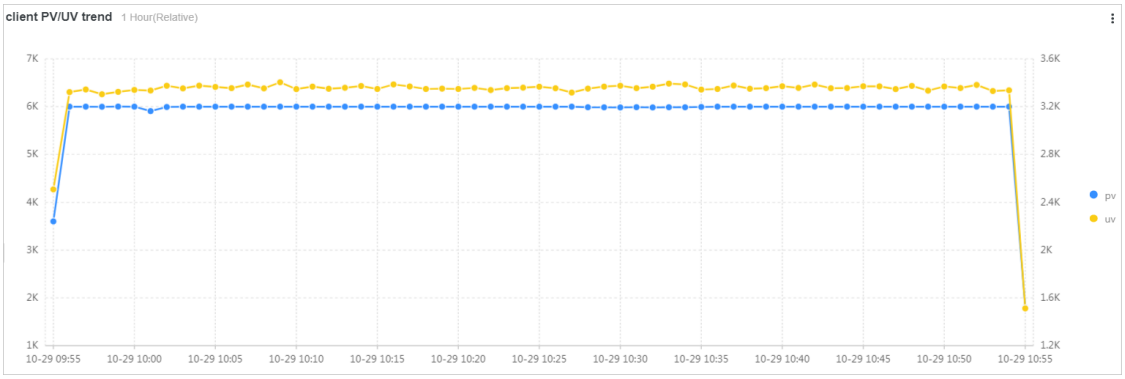
   The destination dashboard includes slb-user-log-slb_layer7_operation_center_en and slb-user-log-slb_layer7_access_center_en.

   - Basic analysis

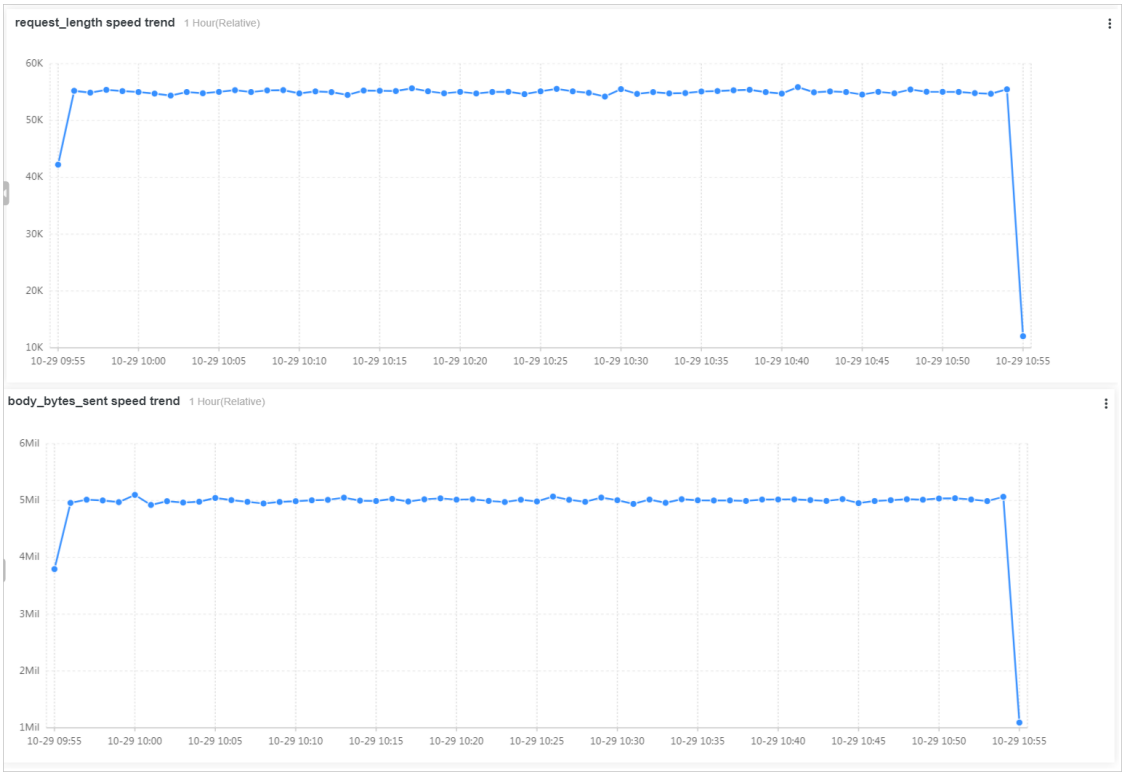
- View the regions from which the requests are sent in a specified period.



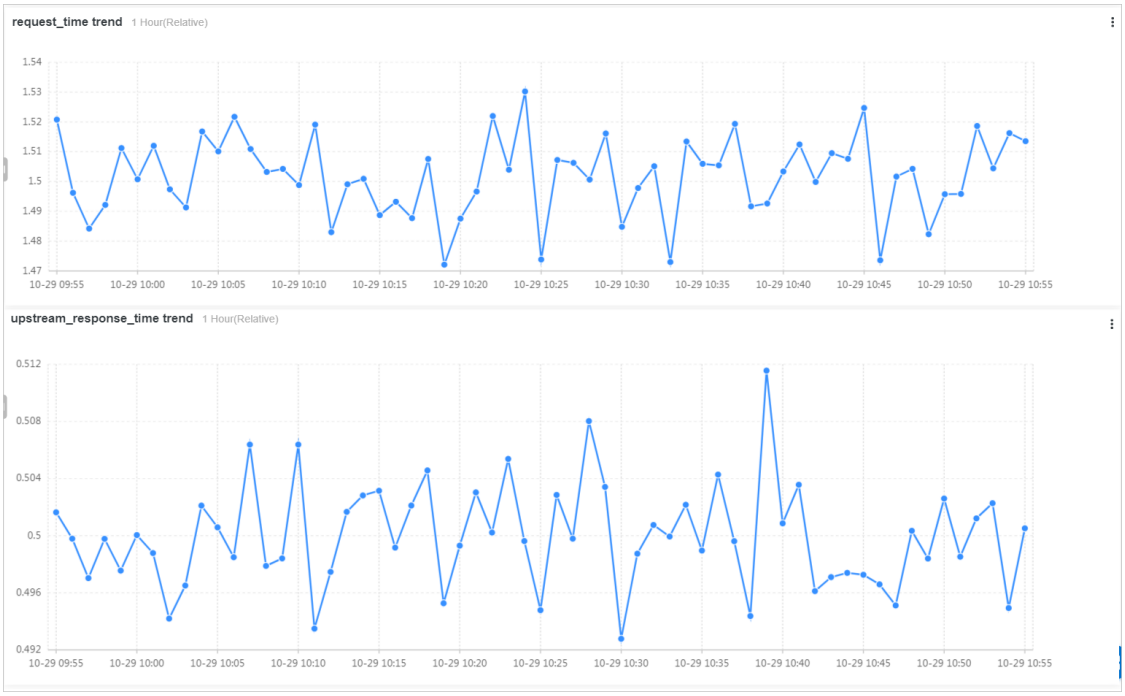
- Use the filter to specify an SLB instance and view the PV and UV trends of the SLB instance. For more information about how to use the filter, see Add a filter.



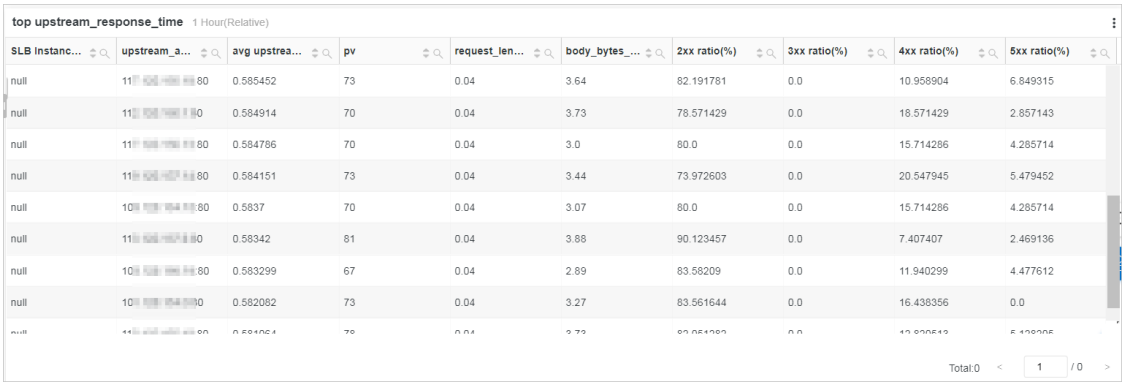
- Traffic and latency analysis
  - View the trends of the request traffic and response traffic in a specified period.



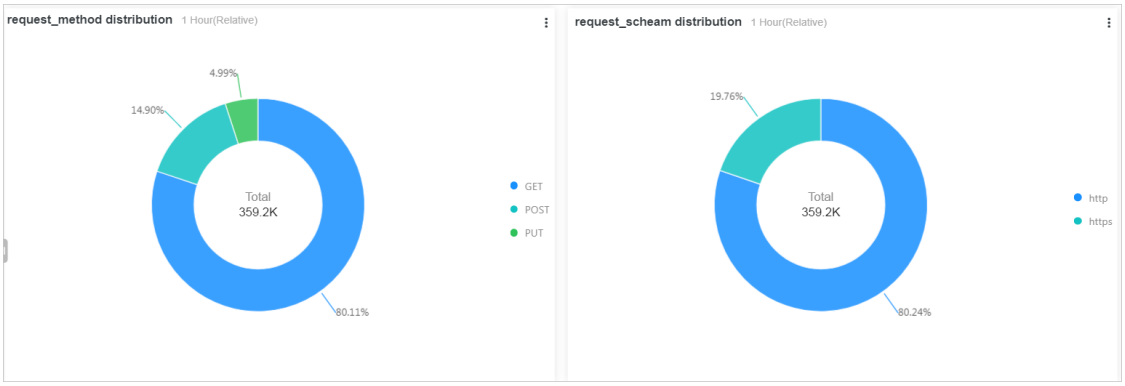
- View the trends of the response time and upstream response time in a specified period.



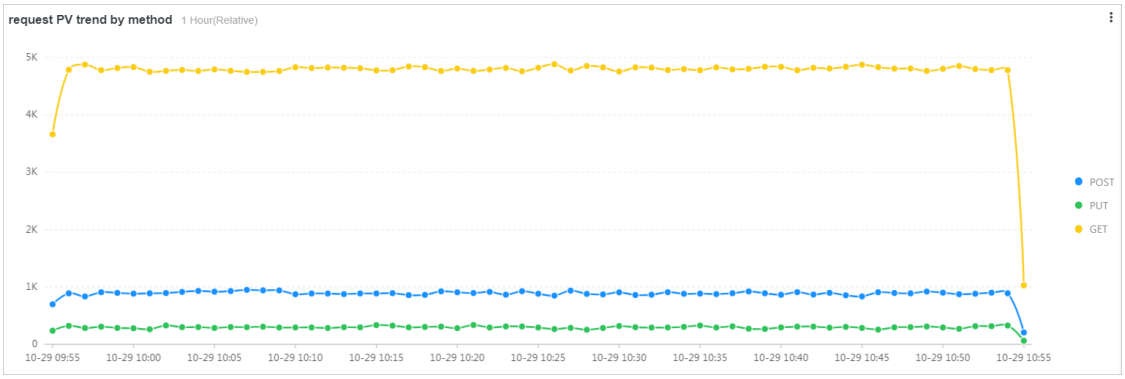
- View the trend of high-latency requests in a specified period.



- User request analysis
  - View the distributions of the request methods and request protocols in a specified period.

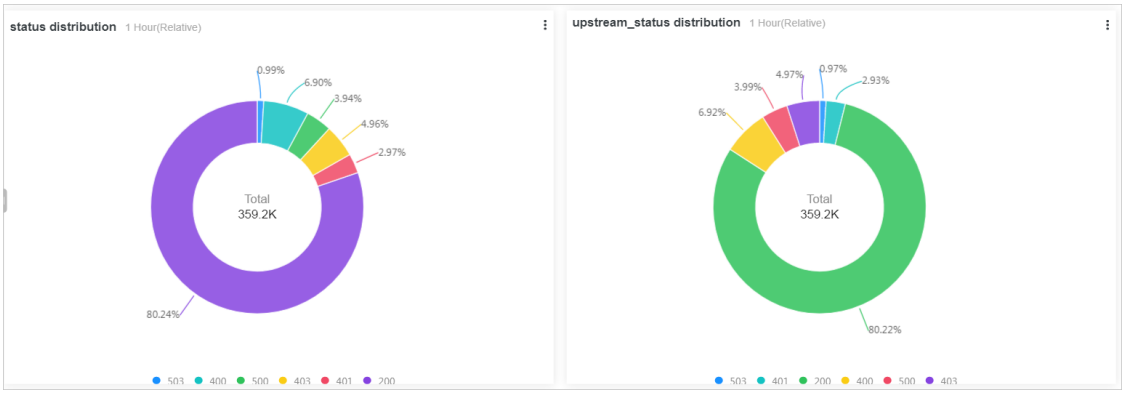


- View the PV trend of different request methods in a specified period.

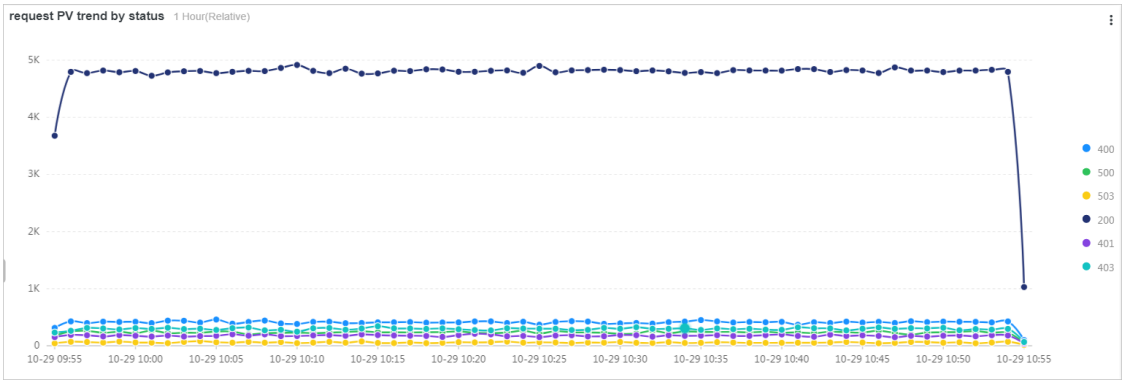


- View the distribution of different status codes in a specified period.

  If a large number of status codes 500 are generated, it indicates that internal errors have occurred on the ECS instance.

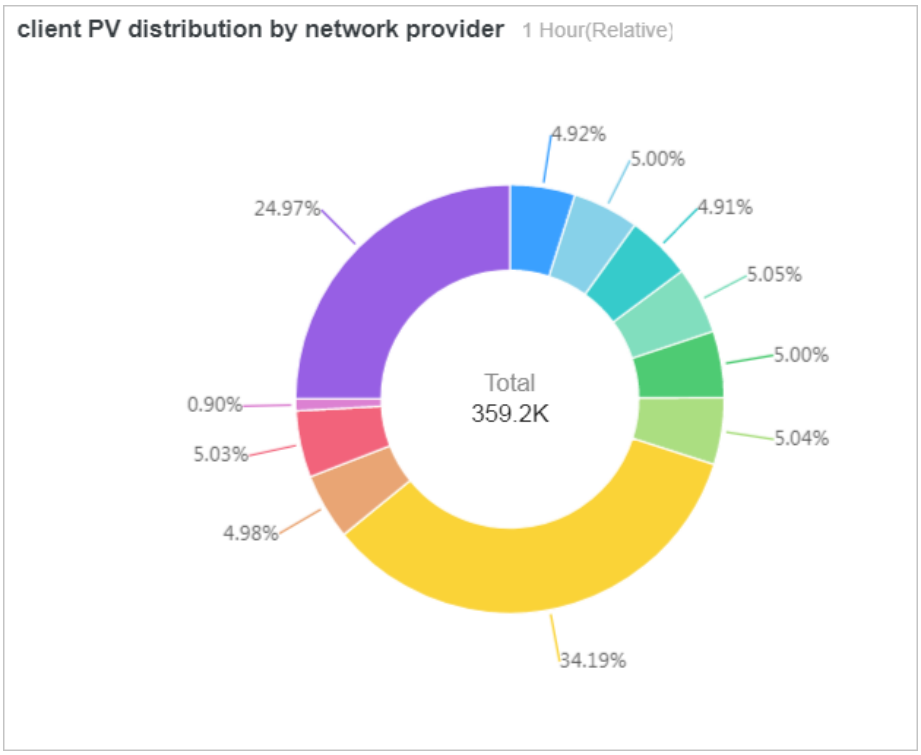


- View the PV trends of different status codes in a specified period.



  - Request source analysis

- View the distribution of Internet service providers.



- View the geographic location such as country, province, and city by analyzing the IP addresses of the clients.

top client 1 Hour(Relative)

client_ip	pv	province	city	provider	request_length(MB)	body_bytes_sent(MB)
1[redacted]9	104	KR_13	Yongin		0.06	4.95

- View the information of a user agent.

  The user agent (http_user_agent) can be used to identify who is visiting the website or service. For example, a search engine uses web crawlers to scan or download website resources. In most cases, web crawlers allow the search engine to update website content in a timely manner and facilitates website promotion and search engine optimization (SEO). If all of the high PV requests are sent from the web crawlers, the service performance may be affected and ECS instance resources may be wasted.

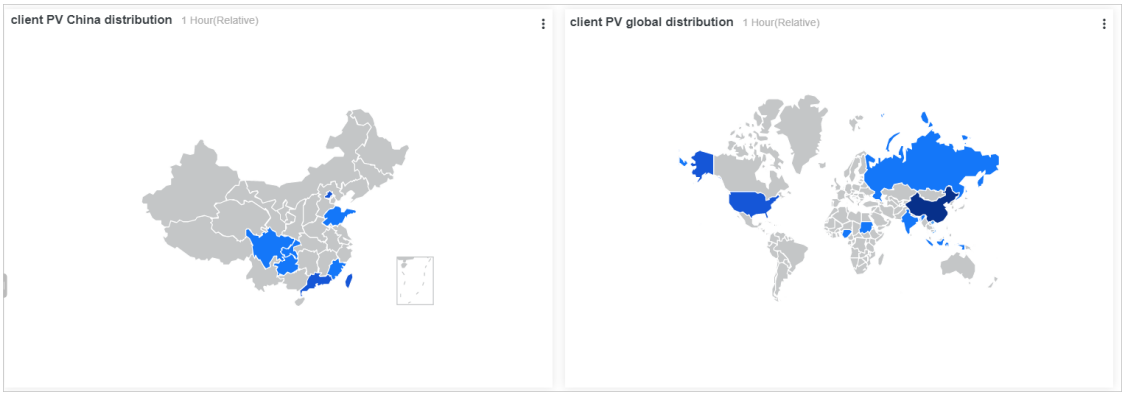
top http_user_agent 1 Hour(Relative)

http_user_agent	pv	request_length(MB)	body_bytes_sent(...	2xx ratio(%)	3xx ratio(%)	4xx ratio(%)	5xx ratio(%)
Mozilla/5.0 (compatible; DotBot/1.1; http://www.opensiteexplorer.org/dotbot, help@moz.com)	146110	76.73	6986.67	80.176579	0.0	14.904524	4.918897
Apache-HttpClient/UNAVAILABLE (java 1.4)	71545	37.56	3401.32	80.310294	0.0	14.740373	4.949333
TurnitinBot (https://turnitin.com/robot/crawlerinfo.html)	71084	37.32	3393.06	80.05599	0.0	14.928817	5.015193
-	35765	18.78	1708.07	80.520062	0.0	14.653991	4.825947
Microsoft Internet Explorer	35196	18.5	1681.23	80.46653	0.0	14.723264	4.810206

- Business analysis

  You can use access logs to analyze service traffic and make business decisions.

- Analyze the PV heat map to optimize promotion plans.



- Analyze visitor behavior based on the host and URI information to optimize website content.

top host 1 Hour(Relative)

SLB instanceID	host	pv	request_length...	body_bytes_se...	2xx ratio(%)	3xx ratio(%)	4xx ratio(%)	5xx ratio(%)
null	www.bd.mock-domain.com	11375	5.94	541.2	80.887912	0.0	14.285714	4.826374
null	www.ac.mock-domain.com	11288	5.89	538.27	80.297661	0.0	14.81219	4.890149
null	www.cb.mock-domain.com	11276	5.9	538.76	79.904221	0.0	15.227031	4.868748
null	www.cd.mock-domain.com	11259	5.9	540.15	80.024869	0.0	14.672706	5.302425
null	www.da.mock-domain.com	11251	5.91	537.98	80.686161	0.0	14.425384	4.888454
null	www.ab.mock-domain.com	11247	5.88	533.82	80.777096	0.0	14.492754	4.73015

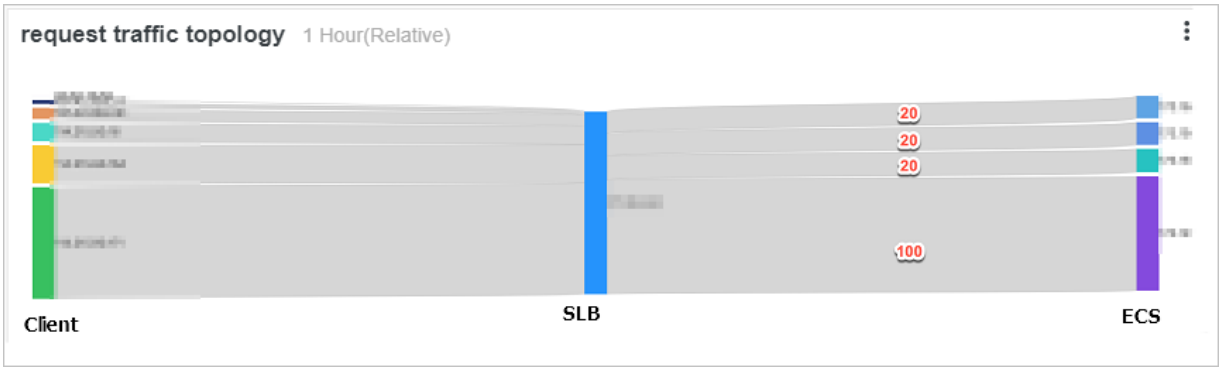
Request scheduling analysis

Client traffic is first processed by SLB and then distributed to an ECS instance for business logic processing. SLB can detect unhealthy ECS instances and distribute traffic to healthy ECS instances. After the unhealthy ECS instances recover, traffic is distributed to them.

Add a listener to the SLB instance to listen to four ECS instances. If an ECS instance (192.168.0.0) acts as a jump server, its performance is four times higher than that of the other three ECS instances. Set the weight of the jumper server (ECS instance) to 100 and set the weight of the other three ECS instances to 20. Run the following query statement to analyze the distribution of request traffic:

```
* | select COALESCE(client_ip, vip_addr, upstream_addr) as source, COALESCE(upstream_addr, vip_addr, client_ip) as dest, sum(request_length) as inflow group by grouping sets( (client_ip, vip_addr), (vip_addr, upstream_addr))
```

The Sankey diagram shows the load of the four ECS instances. After SLB receives traffic, the traffic is distributed to the four ECS instances based on their respective weights (20, 20, 20, and 100).



13.13. Paged query

If a query statement returns a large number of query and analysis results, the results are displayed at a lower speed. Log Service provides the paged query feature to limit the number of logs that can be returned for each query. This topic describes the paging methods of query and analysis results.

Paging methods

Log Service provides the query and analysis feature that allows you to execute a query statement to query logs by using keywords and analyze the query results by using SQL syntax. You can also call the GetLogs operation to query the raw data of logs by using keywords and analyze the query results by using SQL syntax. A query statement can contain a search statement and an analytic statement. The paging methods vary between the search statement and analytic statement. For more information, see GetLogs.

- Search statement: queries the raw data of logs by using keywords. You can configure the offset and line parameters in the GetLogs operation to perform a paged query. For more information, see Query statements.
- Analytic statement: analyzes query results by using SQL syntax. You can use the LIMIT clause to perform a paged query. For more information, see Analytic statements and LIMIT clause.

Paging of query results

The following list describes the offset and line parameters in the GetLogs operation:

- offset: the line from which query results are returned.
- line: the number of lines that are returned for the current API request. A maximum of 100 lines can be returned. If you set this parameter to a value greater than 100, only 100 lines are returned.

When a paged query is performed, the value of the offset parameter increases until all logs are read. When the value reaches a specific number, 0 is returned, and the progress is complete. In this case, all the required data is read.

- Sample code for paging implementation

```
offset = 0                          // Read logs from line 0.
line = 100                         // Read 100 lines at a time.
query = "status:200"                // Read the logs whose value of the status field is 200.
while True:
     response = get_logstore_logs(query, offset, line) // Call the operation to read logs.
     process (response)                                 // Call custom logic to process the return
ed result.
     if response.get_count() == 0 && response.is_complete()
         The read process is complete, and the current loop ends.
     else
        offset += 100                        // The value of the offset parameter increases to 1
00. The next 100 lines are read.
```

- Python code example

  For more information, see Overview.

```
    endpoint = ''      // The Log Service endpoint. For more information, see Endpoints.
    accessKeyId = ''   // The AccessKey ID of your Alibaba Cloud account. For more information, se
e AccessKey pair. An Alibaba Cloud account has permissions to call all API operations. If you use
the AccessKey pair of an Alibaba Cloud account, security risks may occur. We recommend that you cr
eate and use a RAM user to call API operations or perform routine O&M.
    accessKey = ''     // The AccessKey secret of your Alibaba Cloud account.
    project = ''       // The name of the project.
    logstore = ''      // The name of the Logstore.
    client = LogClient(endpoint, accessKeyId, accessKey)
    topic = ""
    From = int(time.time()) - 600
    To = int(time.time())
    log_line = 100
    offset = 0
    while True:
        res4 = None
    for retry_time in range(0, 3):
            req4 = GetLogsRequest(project, logstore, From, To, topic=topic, line=log_line, offset=
offset)
            res4 = self.client.get_logs(req4)
            if res4 is not None and res4.is_completed():
                break
            time.sleep(1)
            offset += 100
            if res4.is_completed() and res4.get_count() == 0:
              break;
            if res4 is not None:
            res4.log_print()  // Display the execution result.
```

- Java code example

  For more information, see Overview.

```
        int log_offset = 0;
        int log_line = 100;   // The number of lines that are read at a time. A maximum of 100 lin
es can be returned. If you want to read more than 100 lines, use the offset parameter. The offset
and line parameters take effect only for a search statement that uses keywords. If you use an anal
ytic statement, these parameters are invalid. If you want an analytic statement to return more tha
n 100 lines, use the LIMIT clause.
        while (true) {
            GetLogsResponse res4 = null;
            // For each offset, 100 lines are read at a time. If the read operation fails, a maxim
um of three retries are allowed.
            for (int retry_time = 0; retry_time < 3; retry_time++) {
                GetLogsRequest req4 = new GetLogsRequest(project, logstore, from, to, topic, query
, log_offset,
                        log_line, false);
                res4 = client.GetLogs(req4);
                if (res4 != null && res4.IsCompleted()) {
                    break;
                }
                Thread.sleep(200);
            }
            System.out.println("Read log count:" + String.valueOf(res4.GetCount()));
            log_offset += log_line;
            if (res4.IsCompleted() && res4.GetCount() == 0) {
                        break;
            }
        }
```

Paging of analysis results

You can use the LIMIT clause for the paging of analysis results. Example:

```
limit Offset, Line
```

The following list describes the offset and line parameters:

- offset: the line from which analysis results are returned.
- line: the number of lines that are returned for the current API request. A maximum of 1,000,000 lines can be returned. If a large number of lines are read at a time, the network delay increases, and the client-side processing speed decreases.

For example, if you want to use the `* | select count(1) , url group by url` statement to return 2,000 lines, you can execute the following statements to perform 4 paged queries and query 500 lines each time:

```
* | select count(1) , url  group by url  limit 0, 500
* | select count(1) , url  group by url  limit 500, 500
* | select count(1) , url  group by url  limit 1000, 500
* | select count(1) , url  group by url  limit 1500, 500
```

- Sample code for paging implementation

```
    offset = 0     // Read logs from line 0.
    line = 500    // Read 500 lines at a time.
    query = "* | select count(1) , url  group by url  limit "
    while True:
    real_query = query + offset + "," +  lines
    response = get_logstore_logs(real_query) // Call the operation to read logs.
    process (response)                       // Call custom logic to process the returned result.
    if response.get_count() == 0
         The read process is complete, and the current loop ends.
    else
        offset += 500                        // The value of the offset parameter increases to 500
. The next 500 lines are read.
```

- Python code example

  For more information, see Overview.

```
    endpoint = ''      // The Log Service endpoint. For more information, see Endpoints.
    accessKeyId = ''   // The AccessKey ID of your Alibaba Cloud account. For more information, se
e AccessKey pair. An Alibaba Cloud account has permissions to call all API operations. If you use
the AccessKey pair of an Alibaba Cloud account, security risks may occur. We recommend that you cr
eate and use a RAM user to call API operations or perform routine O&M.
    accessKey = ''     // The AccessKey secret of your Alibaba Cloud account.
    project = ''       // The name of the project.
    logstore = ''      // The name of the Logstore.
    client = LogClient(endpoint, accessKeyId, accessKey)
    topic = ""
    origin_query = "* | select *  limit "
    From = int(time.time()) - 600
    To = int(time.time())
    log_line = 100
    offset = 0
    while True:
        res4 = None
        query = origin_query + str(offset) + " , " + str(log_line)
        for retry_time in range(0, 3):
            req4 = GetLogsRequest(project, logstore, From, To, topic=topic, query)
            res4 = self.client.get_logs(req4)
            if res4 is not None and res4.is_completed():
                break
            time.sleep(1)
            offset += 100
            if res4.is_completed() and res4.get_count() == 0:
              break;
            if res4 is not None:
            res4.log_print()  // Display the execution result.
```

- Java code example

  For more information, see Overview.

```
        int log_offset = 0;
        int log_line = 500;
        String origin_query = "* | select count(1) , url  group by url  limit "
        while (true) {
            GetLogsResponse res4 = null;
            // For each offset, 500 lines are read at a time. If the read operation fails, a maxim
um of three retries are allowed.
            query = origin_query + log_offset + "," + log_line;
            for (int retry_time = 0; retry_time < 3; retry_time++) {
                GetLogsRequest req4 = new GetLogsRequest(project, logstore, from, to, topic, query
);
                res4 = client.GetLogs(req4);
                if (res4 != null && res4.IsCompleted()) {
                    break;
                }
                Thread.sleep(200);
            }
            System.out.println("Read log count:" + String.valueOf(res4.GetCount()));
            log_offset += log_line;
            if (res4.GetCount() == 0) {
                        break;
            }
        }
```

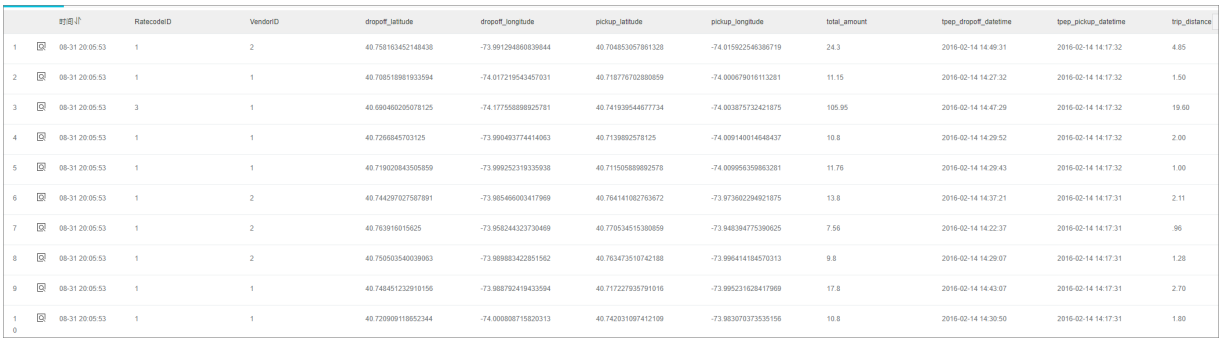
13.14. Analyze vehicle track logs

Taxi companies store trip logs in Alibaba Cloud Log Service and mine useful information based on reliable storage and rapid statistical calculations. This topic describes how taxi companies mine useful information from the data stored in Alibaba Cloud Log Service.

Taxi companies record details for each trip including the time when a passenger gets in and out, latitude and longitude, distance of the trip, payment method, payment amount, and tax amount. Detailed data greatly facilitates the operation of taxi companies. For example, the companies can determine the running intervals in peak hours and dispatch more vehicles to the areas where more taxis are needed. With the help of the data, the requirement of passengers can be met in a timely manner and drivers can have higher incomes. This improves the efficiency of the whole society.

Sample data:

```
RatecodeID:  1VendorID:  2__source__:  192.0.2.1    __topic__:  dropoff_latitude:  40.743995666503906
dropoff_longitude:  -73.983505249023437extra:  0    fare_amount:  9    improvement_surcharge:  0.3
mta_tax:  0.5    passenger_count:  2    payment_type:  1    pickup_latitude:  40.761466979980469    p
ickup_longitude:  -73.96246337890625    store_and_fwd_flag:  N    tip_amount:  1.96    tolls_amount:
0    total_amount:  11.76    tpep_dropoff_datetime:  2016-02-14 11:03:13    tpep_dropoff_time:  14554
18993    tpep_pickup_datetime:  2016-02-14 10:53:57    tpep_pickup_time:  1455418437    trip_distance
:  2.02
```



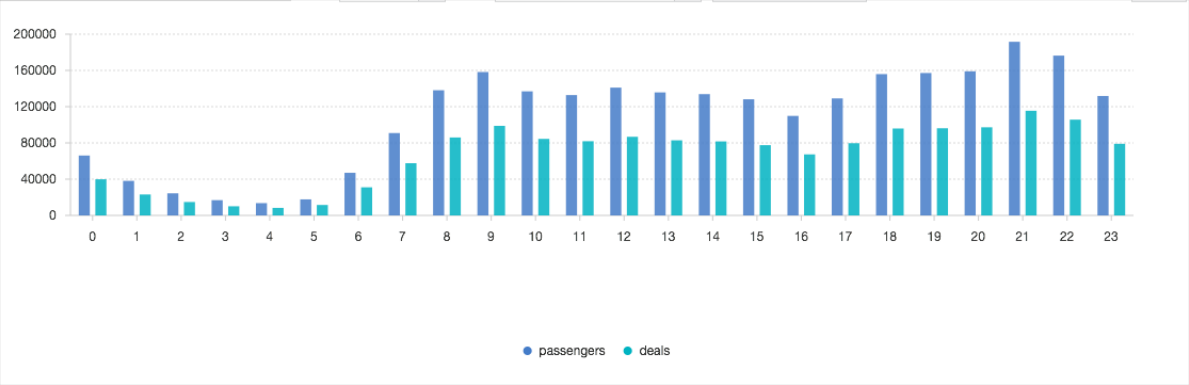
	时间	RatecodeID	VendorID	dropoff_latitude	dropoff_longitude	pickup_latitude	pickup_longitude	total_amount	tpep_dropoff_datetime	tpep_pickup_datetime	trip_distance
1	08-31 20:05:53	1	2	40.758163452148438	-73.991294860839844	40.704853057861328	-74.015922546386719	24.3	2016-02-14 14:49:31	2016-02-14 14:17:32	4.85
2	08-31 20:05:53	1	1	40.708518981933594	-74.017219543457031	40.718776702880859	-74.000679016113281	11.15	2016-02-14 14:27:32	2016-02-14 14:17:32	1.50
3	08-31 20:05:53	3	1	40.690460205078125	-74.177558898925781	40.741939544677734	-74.003875732421875	105.95	2016-02-14 14:47:29	2016-02-14 14:17:32	19.60
4	08-31 20:05:53	1	1	40.7266845703125	-73.990493774414063	40.7139892578125	-74.009140014648437	10.8	2016-02-14 14:29:52	2016-02-14 14:17:32	2.00
5	08-31 20:05:53	1	1	40.719020843505859	-73.999252319335938	40.711505889892578	-74.009956359863281	11.76	2016-02-14 14:29:43	2016-02-14 14:17:32	1.00
6	08-31 20:05:53	1	2	40.744297027587891	-73.985466003417969	40.764141082763672	-73.973602294921875	13.8	2016-02-14 14:37:21	2016-02-14 14:17:31	2.11
7	08-31 20:05:53	1	2	40.76391601562	-73.958244323730469	40.770534515380859	-73.948394775390625	7.56	2016-02-14 14:22:37	2016-02-14 14:17:31	.96
8	08-31 20:05:53	1	2	40.750503540039063	-73.989883422851562	40.763473510742188	-73.996414184570313	9.8	2016-02-14 14:29:07	2016-02-14 14:17:31	1.28
9	08-31 20:05:53	1	1	40.748451232910156	-73.988792419433594	40.717227935791016	-73.995231628417969	17.8	2016-02-14 14:43:07	2016-02-14 14:17:31	2.70
10	08-31 20:05:53	1	1	40.720909118652344	-74.000808715820313	40.742031097412109	-73.983070373535156	10.8	2016-02-14 14:30:50	2016-02-14 14:17:31	1.80

Common statistics

Before query and analysis, you must enable and configure the index feature. For more information, see Configure indexes.

- Run the following statement to count the number of passengers boarding taxis during the day and determine the peak hours:

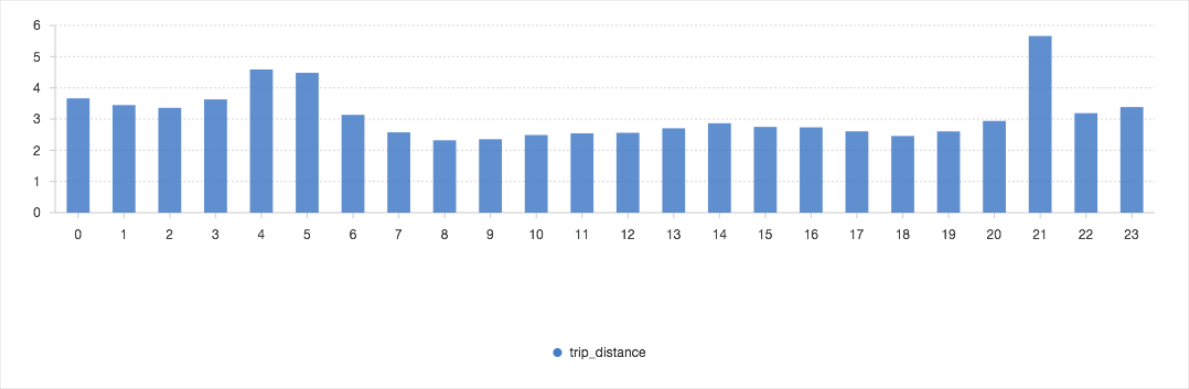
```
*| select count(1) as deals, sum(passenger_count) as passengers,
(tpep_pickup_time %(24*3600)/3600+8)%24 as time
group by (tpep_pickup_time %(24*3600)/3600+8)%24 order by time limit 24
```



As shown in the preceding figure, the peak hours are generally the morning hours when people go to work and the evening hours when people get off work. Based on this data, taxi companies can dispatch more vehicles accordingly.

- Run the following statement to collect statistics about the average trip distance in different time periods:

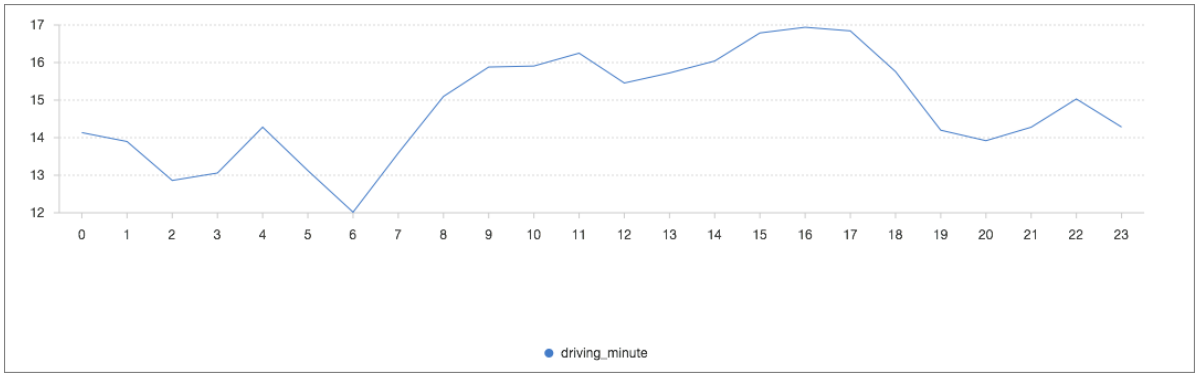
```
*| select  avg(trip_distance)  as trip_distance,
(tpep_pickup_time %(24*3600)/3600+8)%24 as time
group by (tpep_pickup_time %(24*3600)/3600+8)%24 order by time limit 24
```



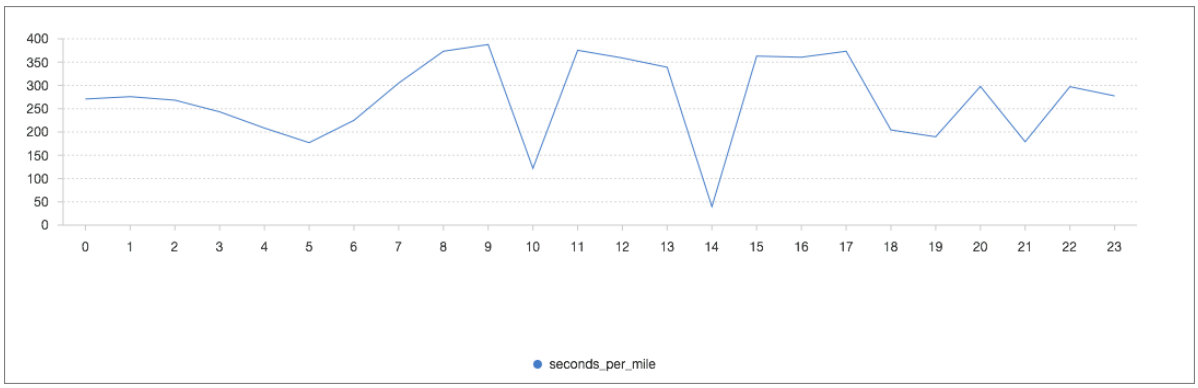
Passengers tend to take a longer trip during certain time periods of the day, so taxi companies need to dispatch more vehicles.

- Run the following statement to calculate the average trip duration (in minutes) and the time required for per unit of mileage (in seconds), and determine during which time period of the day taxis experience more traffic:

```
*| select  avg(tpep_dropoff_time-tpep_pickup_time)/60  as driving_minutes,
(tpep_pickup_time %(24*3600)/3600+8)%24 as time
group by (tpep_pickup_time %(24*3600)/3600+8)%24 order by time limit 24
```



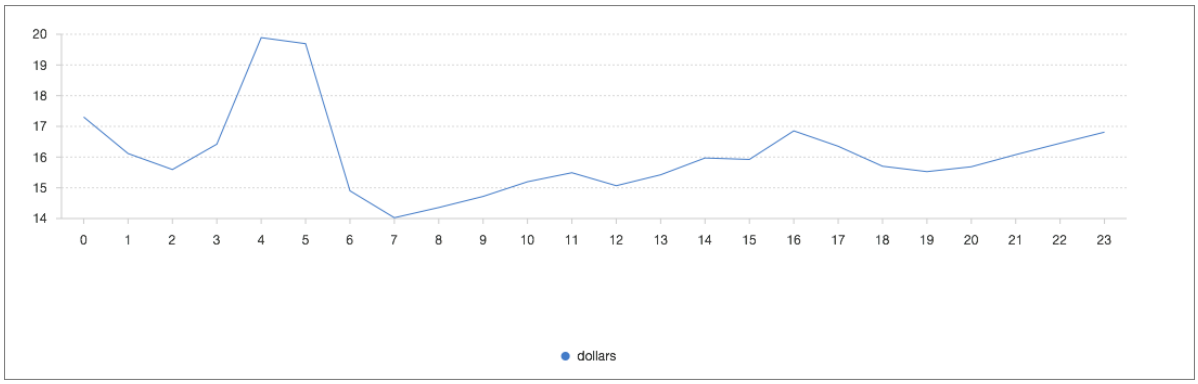

```
*| select  sum(tpep_dropoff_time-tpep_pickup_time)/sum(trip_distance)  as driving_minutes,
(tpep_pickup_time %(24*3600)/3600+8)%24 as time
group by (tpep_pickup_time %(24*3600)/3600+8)%24 order by time limit 24
```



More vehicles must be dispatched during peak hours.

- Run the following statement to calculate average taxi fares during different time periods and determine the hours with more income:

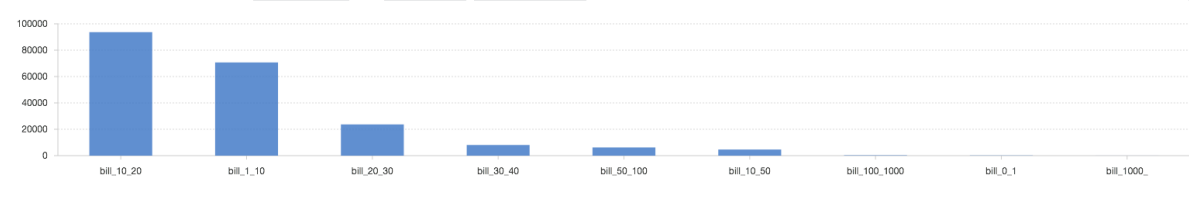
```
*| select  avg(total_amount)  as dollars,
(tpep_pickup_time %(24*3600)/3600+8)%24 as time
group by (tpep_pickup_time %(24*3600)/3600+8)%24 order by time limit 24
```



The average taxi fares per customer are higher around 4:00 AM, so financially challenged drivers can consider providing services during this time period.

- Run the following statement to view the distribution of payment amounts:

```
*| select case when total_amount < 1 then 'bill_0_1'
when total_amount < 10 then 'bill_1_10'
when total_amount < 20 then 'bill_10_20'
when total_amount < 30 then 'bill_20_30'
when total_amount < 40 then 'bill_30_40'
when total_amount < 50 then 'bill_10_50'
when total_amount < 100 then 'bill_50_100'
when total_amount < 1000 then 'bill_100_1000'
else 'bill_1000_'  end
as bill_level , count(1) as count group by
case when total_amount < 1 then 'bill_0_1'
when total_amount < 10 then 'bill_1_10'
when total_amount < 20 then 'bill_10_20'
when total_amount < 30 then 'bill_20_30'
when total_amount < 40 then 'bill_30_40'
when total_amount < 50 then 'bill_10_50'
when total_amount < 100 then 'bill_50_100'
when total_amount < 1000 then 'bill_100_1000'
else 'bill_1000_'  end
order by count desc
```



As shown in the preceding figure, the payment amount of most transactions ranges from USD 1 to USD 20.

13.15. Analyze sales system logs

Bills are the core data of e-commerce companies, and the outcome of a series of marketing and promotional activities. Billing data contains a lot of valuable information. The information helps you define user profiles and create guidelines for future marketing plans. The billing data can also serve as an indicator of popularity and used to provide suggestions for subsequent stocking options.

Billing data is stored as logs in Alibaba Cloud Log Service. With the computing capacity to process hundreds of millions of log entries per second, Log Service supports high-speed queries and SQL-based statistics. This topic uses several examples to describe how to mine useful information from billing data.

The following example uses a complete bill that contains goods information (name and price), deal information (final price, payment method, and discount information), and buyer information (membership information):

```
__source__:  10.164.232.105      __topic__:  bonus_discount:  category:  men's clothing   commodity:
Everyday discount autumn and winter teenager velvet and thickened skinny jeans men's winter slim pant
s commodity_id:  443 discount:  member_discount:  member_level:  nomember_point:  memberid:  mobile:
pay_transaction_id:  060f0e0d080e0b05060307010c0f0209010e0e010c0a0605000606050b0c0400 pay_with:  alip
ay real_price:  52.0 suggest_price:  52.0
```

Perform statistical analysis

Before query and analysis, enable and configure the index feature. For more information, see Configure indexes.

- View the percentage of the sales of each category of products to the total sales.

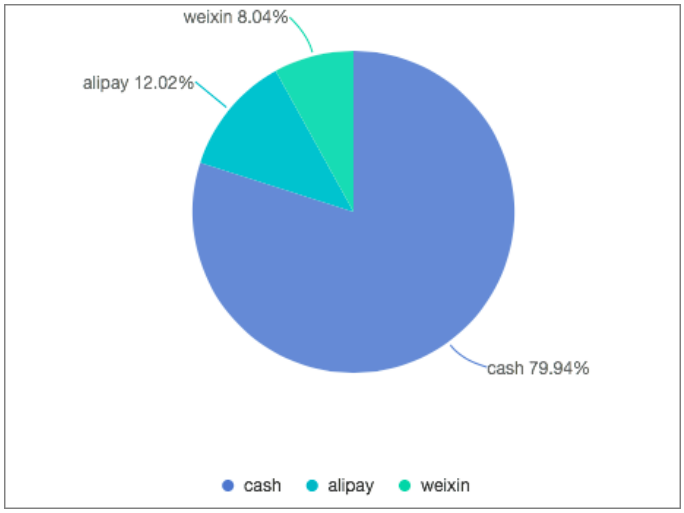
  ```
  *|select count(1) as pv ,category group by category limit 100
  ```

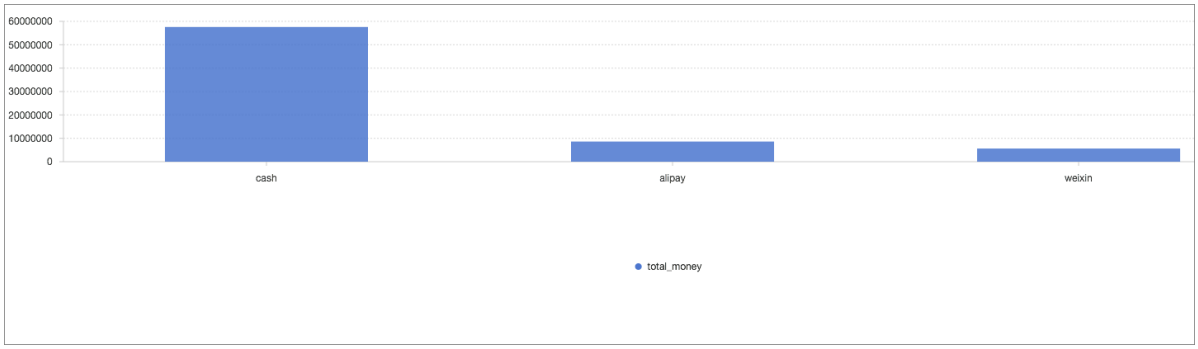
- View the sales trends of different women's clothes.

```
category: women's clothing  | select count(1) as deals , commodity
group by commodity order by deals  desc limit 20
```

- View share and turnover of different payment methods.

```
*  | select count(1) as deals , pay_with group by pay_with  order by deals  desc limit 20
*  | select sum(real_price)  as total_money , pay_with group by pay_with  order by total_money  de
sc limit 20
```





14.FAQ

14.1. FAQ about query and analysis

This topic lists some frequently asked questions about the query and analysis feature of Log Service.

14.2. FAQ about log query

This topic describes the FAQ about log query.

How do I identify the source server from which Logtail collects logs during a query?

If a machine group uses IP addresses as its identifier when logs are collected by using Logtail, servers in the machine group are distinguished by internal IP addresses. When you query logs, you can use the hostname and custom IP address to identify the source server from which logs are collected.

For example, you can use the following statement to count the times different hostnames appear in logs:

> **Note** You must configure an index for the __tag__:__hostname__ field and enable the statistics feature.

```
* | select "__tag__:__hostname__" , count(1) as count group by "__tag__:__hostname__"
```

__tag__:__hostname__	count
logtail-ds-vht2d	4255
logtail-ds-9nrxb	2177
null	8
logtail-ds-wghkv	2625

How do I query IP addresses in logs?

You can use the exact match method to query IP addresses in logs. You can search for log data by IP address. For example, you can specify whether to include or exclude an IP address. However, you cannot use the partial match method to query log data. This is because decimal points contained in an IP address are not default delimiters in Log Service. You can also filter data by using other methods. For example, you can use an SDK to download data and then use a regular expression or the string.indexof() method to search for results.

For example, if you execute the following statement, the log entries that are retrieved from the 121.42.0 CIDR block are still returned.

```
not ip:121.42.0 not status:200 not 360jk not DNSPod-Monitor not status:302 not jiankongbao
        not 301 and status:403
```

How do I use two conditions to query log data?

If you need to use two conditions to query logs, enter two statements at the same time.

For example, you want to query log entries whose status field is neither OK nor Unknown in a Logstore. You can use the `not OK not Unknown` statement to retrieve expected results.

How can I query collected logs in Log Service?

You can use one of the following methods to query logs in Log Service:

1. Use the Log Service console. For more information, see Query and analyze logs.
2. Use an SDK. For more information, see SDK overview.
3. Use the Restful API. For more information, see GetLogs.

14.3. What do I do if no results are returned when I query a log?

This topic describes how to troubleshoot the issue that no results are returned when you query a log in the Log Service console.

Log collection failures

If the log is not collected to Log Service, no results are returned when you query the log. You can check whether the log appears in the Preview Data section of the Configure Query and Analysis step. If yes, the log is collected to Log Service. We recommend that you proceed with troubleshooting based on other possible reasons. If no, proceed with troubleshooting based on the following reasons:

- The log is not generated in the log source.

  Check your log source.

- Logtail has no heartbeats.

  On the **Machine Group Status** page, check whether Logtail that is installed on your server has a heartbeat. If Logtail has no heartbeats, see What do I do if no heartbeat connections are detected on Logtail?

- The log file from which logs are collected has no data written in real time.

  Open the */usr/local/ilogtail/ilogtail.LOG* file to view detailed error information. The following list describes common error messages:

  - parse delimiter log fail: The error message returned because an error occurs when Log Service collects logs in delimiter mode.
  - parse regex log fail: The error message returned because an error occurs when Log Service collects logs in full regex mode.

Delimiter configuration errors

You can view the delimiter that you specify, use the delimiter to split a log, and then check whether you can obtain a keyword based on which you can query the log. For example, the delimiter `,;=()[]{}?@&<>/:'` is used to split the log `abc"defg,hij`, and `abc"defg` and `hij` are obtained. If you use `hij` to query the log, the log is returned. If you use `abc` to query the log, no results are returned.

Note

- To reduce your index costs, we recommend that you use field indexes. For more information, see Index types.
- The indexing feature takes effect only on the log data that is written after you configure indexes. If you want to query and analyze historical data, you can use the reindexing feature. For more information, see Reindex logs for a Logstore.

  In the **Search & Analysis** panel, you can check whether the delimiter that you specify meets the requirements. For more information, see Configure indexes.

Other reasons

- Check whether the time of the log falls in the time range of your query.
- Log Service allows you to preview logs in real time, but the query feature may cause a maximum of 1-minute latency. We recommend that you query logs at least 1 minute after the logs are generated.

If the issue persists, submit a ticket.

14.4. What can I do if the "The results are inaccurate" error occurs?

This topic describes how to resolve the **The results are inaccurate** error that may occur when you query logs.

Issue

If the **The results are inaccurate** error occurs when you query logs, Log Service fails to scan all log data and the returned results are inaccurate.

Cause

The error may occur due to the following causes:

- The time range specified for the query is too large.

  For example, if the specified time range is three months or one year, Log Service cannot scan all data of this time period. Only partial results are returned. Therefore, data accuracy is compromised.
- The conditions specified in the search statement are too complicated.

  For example, if you specify 30 conditions in a query statement, Log Service cannot read all log data.
- The amount of data to be read is too large.

  If you specify multiple fields in an analytic statement, and the data volume to be read exceeds the read capacity of a shard, the returned results are inaccurate. This is because each shard can read only 1 GB of data.

Solution

Narrow down the query time range and perform multiple queries (up to 10 queries) to get the complete results.

14.5. How do I resolve common errors that occur when I query and analyze log data?

This topic describes the common error messages that are returned when you query and analyze log data in the Log Service console and provides solutions to the errors.

line 1:44: Column 'XXX' cannot be resolved;please add the column in the index attribute

- Cause

  No index is configured for the XXX field.

- Solution

  Configure an index and enable the analysis feature for the field. For more information, see Configure indexes.

ErrorType:QueryParseError.ErrorMessage:syntax error error position is from column:10 to column:11,error near < : >

- Cause

  The syntax of the query statement is invalid. The position of the invalid syntax is near the colon (:).

- Solution

  Check and modify the query statement, and then execute the query statement.

Column 'XXX' not in GROUP BY clause;please add the column in the index attribute

- Cause

  You can specify a GROUP BY clause in a SELECT statement to perform aggregate calculations. In the SELECT statement, you can specify only a column that is specified in the GROUP BY clause or specify a random column that is excluded from the GROUP BY columns as an argument in an aggregate function. For example, `* | SELECT status, request_time, COUNT(*) AS PV GROUP BY status` is an invalid query statement because request_time is not a GROUP BY column.

- Solution

  Check and modify the query statement, and then execute the query statement. In contrast to the preceding query statement, the following query statement is valid: `* | SELECT status, arbitrary(request_time), count(*) AS PV GROUP BY status`. For more information, see GROUP BY clause.

sql query must follow search query,please read syntax doc

- Cause

  The syntax of the query statement is invalid because only an analytic statement is specified. In Log Service, you must specify an analytic statement together with a search statement in the `Search statement|Analytic statement` format.

- Solution

  Add a search statement prior to the analytic statement, for example, `* | SELECT status, count(*) AS PV GROUP BY status`. For more information, see Syntax.

line 1:10: identifiers must not start with a digit; surround the identifier with double quotes

- Cause

  The syntax of the analytic statement is invalid because the column name or variable name that is specified in the analytic statement starts with a digit. A column name in an SQL statement can contain only letters, digits, and underscores (_). The column name must start with a letter.

- Solution

  Change the alias. For more information, see Column aliases.

line 1:9: extraneous input '' expecting

- Cause

  Extra Chinese quotation marks are specified in the query statement.

- Solution

  Check and modify the query statement, and then execute the query statement.

key (XXX) is not config as key value config,if symbol : is in your log,please wrap : with quotation mark "

- Cause

  No index is configured for the XXX field, or the field that you specify contains special characters such as space characters and is not enclosed in double quotation marks ("").

- Solution
  - Configure an index and enable the analysis feature for the field. For more information, see Configure indexes.
  - Enclose the field in double quotation marks ("").

Query exceeded max memory size of 3GB

- Cause

  The size of the memory that is used by the query statement exceeds 3 GB. The issue occurs because a large number of values are returned in the query and analysis result after you use a GROUP BY clause to remove duplicates.

- Solution

  Optimize the GROUP BY clause. Reduce the number of fields that are specified in the GROUP BY clause.

ErrorType:ColumnNotExists.ErrorPosition,line:0,column:1.ErrorMessage:line 1:123: Column 'XXX' cannot be resolved; it seems XXX is wrapper by ";if XXX is a string ,not a key field, please use 'XXX'

- Cause

  XXX is not an indexed field and cannot be enclosed in double quotation marks (""). If you want to use a string in an analytic statement, you must enclose the string in single quotation marks (''). Strings that are not enclosed or enclosed in double quotation marks ("") indicate field names or column names.

- Solution
  - If you want to analyze the XXX field, make sure that you configure an index and enable the analysis feature for the field. For more information, see Configure indexes.
  - If XXX is a string, you must enclose the string in single quotation marks ('').

user can only run 15 query concurrently

- Cause

  More than 15 analytic statements are executed at the same time. Each project supports a maximum of 15 concurrent analytic statements.

- Solution

  Reduce the number of concurrent requests to execute multiple search statements based on your business requirements.

unclosed string quote

- Cause

The double quotation marks (") in the query statement are incomplete.

- Solution

  Check and modify the query statement, and then execute the query statement.

error after :.error detail:error after :.error detail:line 1:147: mismatched input 'in' expecting {<EOF>, 'GROUP', 'ORDER', 'HAVING', 'LIMIT', 'OR', 'AND', 'UNION', 'EXCEPT', 'INTERSECT'}

- Cause

  The invalid keyword in is specified.

- Solution

  Check and modify the query statement, and then execute the query statement.

Duplicate keys (XXX) are not allowed

- Cause

  Duplicate indexes are configured for fields.

- Solution

  Check the index configurations. For more information, see Configure indexes.

only support * or ? in the middle or end of the query

- Cause

  Wildcards are not used in the expected positions when you perform a fuzzy search.

- Solution

  Check and modify the wildcards in the query statement. Take note of the following rules:

  - You can add an asterisk (*) or a question mark (?) as a wildcard to the middle or end of a keyword to perform a fuzzy search.
  - A keyword cannot start with an asterisk (*) or a question mark (?).
  - The long and double data types do not support asterisks (*) or question marks (?) in fuzzy searches.

logstore (xxx) is not found

- Cause

  The XXX Logstore does not exist or no index is configured.

- Solution

  Check whether the Logstore exists. If the Logstore exists, you must configure an index for at least one field and enable the analysis feature for the Logstore.

condition number 43 is more than 30

- Cause

  The number of fields that are specified in the search statement is 43. Only a maximum of 30 fields can be specified in a search statement.

- Solution

  Modify the search statement to reduce the number of specified fields and make sure that the number of fields is less than or equal to 30.

ErrorType:SyntaxError.ErrorPosition,line:1,column:19.ErrorMessage:line 1:19: Expression "data" is not of type ROW

- Cause

  The data type of the field that is specified in the query statement is invalid.

- Solution

  Check and modify the query statement, and then execute the query statement.

14.6. What are the differences between LogHub and LogSearch?

Log Service provides two features that need to read log data: LogHub and LogSearch. Their difference lies in that LogHub provides log collection and distribution channels, whereas LogSearch allows you to query logs.

Differences between LogHub and LogSearch

Both LogHub and LogSearch of Log Service need to read log data:

LogHub: provides public channels for log collection and distribution. It reads and writes full data in first-in, first-out (FIFO) order, which is similar to Kafka.

- Each Logstore has one or more shards. Data is written to a random shard.
- You can read multiple logs at a time from a specified shard based on the order in which the logs were written to the shard.
- You can set the start position (cursor) for pulling logs in shards according to the time when the server receives these logs.

LogSearch: enables you to query and analyze a large number of logs based on LogHub, and set conditions to query and collect statistics on logs.

- LogSearch allows you to search for required data based on query conditions.
- LogSearch supports a Boolean combination of the keywords AND, NOT, and OR, and also supports SQL query statistics.
- LogSearch is independent of shards.

Differences between LogSearch and LogHub

Feature	LogSearch	LogHub
Search by keyword	Supported.	Not supported.
Data read (a small amount of data)	Quick.	Quick.
Data read (full data)	Slow. LogSearch reads 100 logs in 100 ms, so this method is not recommended.	Quick. LogHub reads 1 MB logs in 10 ms, so this method is recommended.
Data read by topic	Yes.	No. Data is identified only by shard.
Data read by shard	No. Data in all shards is queried.	Yes. You need to specify a shard each time to read data.
Price	Relatively high.	Low.
Scenarios	Monitoring, problem investigation, and analysis.	Full data processing scenarios, such as stream computing and batch processing.

14.7. Fuzzy match

This topic describes three methods to implement fuzzy match.

Include wildcard characters in query statements to implement fuzzy match

An asterisk (*) indicates zero or more occurrences of characters. A question mark (?) indicates one occurrence of a character. For example, **abc*** indicates that a word is matched if the word starts with abc. **ab? d** indicates that a word is matched if the word starts with ab, ends with d, and contains one character in between. For more information, see Search syntax.

> **Note** If you use wildcard characters to implement fuzzy match, a maximum of 100 words are matched. The returned data is the log entries that include the matched words. If the prefix is short, the number of matched words may exceed 100. In this case, only a part of matched log entries are returned. In addition, if you combine the NOT clause with wildcard characters, only a part of words can be filtered. For example, if you execute the **not abcd*** statement, words that start with abcd are still returned.

Use the LIKE clause to implement fuzzy match

The LIKE clause complies with the standard SQL LIKE syntax. The percent sign (%) in the LIKE clause indicates zero or more occurrences of characters. The underscore (_) indicates one occurrence of a character.

Example: To query log entries that include fields whose names start with abcd, execute the following statement:

```
* | select * from log where key like 'abcd%'
```

Use regular expression functions to implement fuzzy match

You can specify a regular expression in a regular expression function to match multiple words. Regular expressions can match characters and digits. They can better satisfy your business requirements. For more information, see Regular expression functions.

Examples:

- *** | select * from log where regexp_like(key, abc*)**: returns the words that start with abc.
- *** | select * from log where regexp_like(key, abc\d+)**: returns the words that start with abc. In addition, abc is followed by digits.
- *** | select * from log where regexp_like(key, abc[xyz])**: returns the words that start with abc. In addition, abc is followed by x,y, or z.

14.8. How do I query logs by using exact match?

If you want to query logs by using exact match of multiple keywords, you can use the LIKE clause.

- Sample log

```
body_bytes_sent:1061
http_user_agent:Mozilla/5.0 (Windows; U; Windows NT 5.1; ru-RU) AppleWebKit/533.18.1 (KHTML, like
Gecko) Version/5.0.2 Safari/533.18.5
remote_addr:192.0.2.2
remote_user:vd_yw
request_method:DELETE
request_uri:/request/path-1/file-5
status:207
time_local:10/Jun/2021:19:10:59
```

- Query requirement

  Query the logs whose http_user_agent field value contains the exact phrase `like Gecko`.

- Incorrect query statement

  ```
  "like" and "Gecko"
  ```

  This query returns the logs whose http_user_agent field value contains the following phrases: `like Gecko`, `Gecko like`, `like abc Gecko`, or `Gecko abc like`.

- Correct query statement

  ```
  * | Select * where http_user_agent like '%like Gecko%'
  ```

  The http_user_agent parameter specifies the field based on which the system queries logs.

  The LIKE clause complies with the LIKE syntax in standard SQL. The percent sign (%) in a LIKE clause indicates zero or more occurrences of characters. The underscore (_) indicates one occurrence of a character.