DORIS 文档

2021-03-18

百度智能云

目录

Product Description

Product Introduction

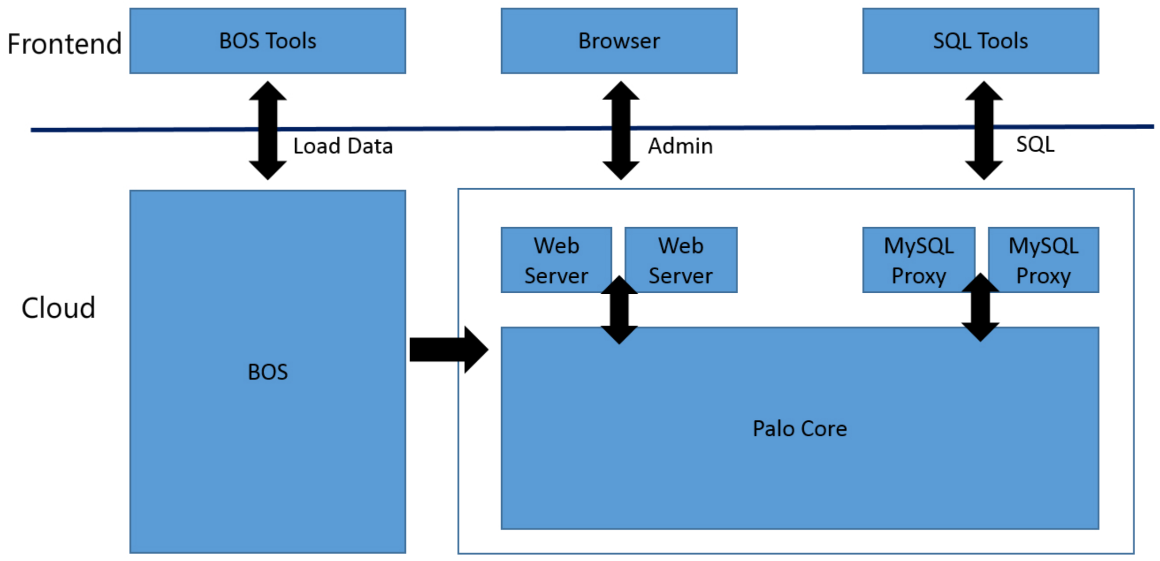
Palo, Baidu Data Warehouse, is a PB-level MPP data warehouse service provided by Baidu AI Cloud. It provides high-performance analysis and report query features on Big Data sets at low cost.

Palo is not an OLTP-oriented database product, but an OLAP-oriented database product. Products with features similar to Palo include commercial data warehouse systems such as Greenplum, Vertica, Exadata and cloud services such as Amazon RedShift and Google BigQuery. You can refer to the above-mentioned products to better understand Palo.

System Architecture

Introduction of System Architecture

Schematic Diagram



External products of Palo, Baidu Data Warehouse, consist of cloud and frontend. The frontend provides users with tools to interact with the cloud, which can upload data to BOS, perform Palo cluster management and submit SQL statements.

Data Import

Palo cloud products currently support only data import from the BOS System. The user needs to use BOS-related tools to upload data to BOS system and import it into Palo.

For private Palo clusters that are not in the cloud, they can be imported from HDFS or the bulk load command can be used to import them from a local file. For detailed information, see the SQL manual.

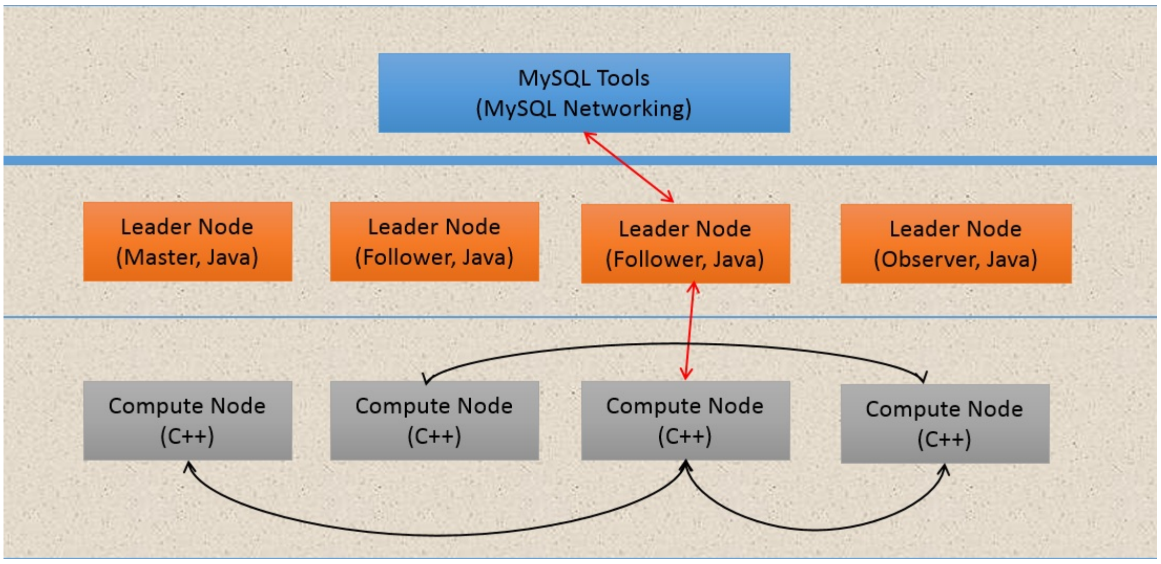
Cluster Management

Palo currently provides a web management interface through which users can complete cluster application, add and delete nodes, view cluster information, reset passwords and perform other operations.

SQL Interaction

Users can be connected to Palo by using any MySQL-connected tool or library. For example, SQL can be submitted by connecting the Palo through MySQL client, JDBC, ODBC and other BI tools, including DDL, DML, DQL, DCL and other statements. The connection url of JDBC and ODBC can be viewed on the cluster management page (see Quick Start).

Palo Core is the core engine of Palo, which is implemented as follows:



Palo Core is mainly divided into two roles, i.e., Leader Node and Compute Node, each of which can be deployed on multiple machines.

Leader Node

Leader Node is responsible for maintaining metadata of the system, receiving all SQL requests from users, analyzing SQL, forming physical execution plan, and then distributing tasks to Compute Node for distributed execution. There is one Master in all Leader Nodes, which is responsible for performing metadata change. Other Leader Nodes are synchronized with Master for their metadata regularly to realize fault tolerance and improve performance.

Compute Node

Compute Node consists of a stand-alone data storage engine and a query task execution engine, and receives task execution requests from the Leader Node. Compute Node is the one with heaviest load in the whole Core system, and completes all storage and computing loads. Palo uses shared nothing architecture, so that Compute Node can achieve linear expansion.

Key Features

MySQL Interface

Through MySQL-compatible interfaces, users can directly use MySQL-related libraries or tools without the deployment of new client libraries or tools. Due to the provision of the MySQL interface, it’s easily compatible with upper-layer applications. And, the learning curve of users is reduced, facilitating users to use them.

High Concurrency and Small Query

Unlike open-source data analysis tools such as Hive, Impala, Palo supports high concurrencies and small queries, with 100 clusters being up to 10w QPS.

Large Query and High Throughput

The throughput of large queries is improved by using Partition Pruning, pre-aggregation, predicate push-down, vectorization execution, and other technologies.

Fault Tolerance and Stability

Both Leader Node and Compute Node provide fault tolerance and ensure the stability of the system. Multiple Leader Nodes provide a high-reliability guarantee for metadata; Multiple replicas of data in Compute Node provide high data reliability.

Efficient Column Type Storage Engine

It has intelligent indexes such as Min/Max, prefix index, etc., and uses an efficient column type storage engine adopting RLE coding.

Efficient Materialized Index Rollup Index

Have the ability to create Rollup Index. A Rollup Index contains some common columns in a Table and is physically stored independently. For the report query with fixed query mode, the query speed can be greatly improved.

Easy to Modify Table Structure

It supports modification of the table structure when data has been imported, including adding columns, deleting columns, modifying column types, and changing column order.

Low Cost

In terms of cost performance, it is 10 to 100 times higher than other commercial products.

Pricing

This document introduces Palo's pricing, billing rules and expiration reminders.

On-demand Billing

In case of on-demand billing, Palo is charged according to the configuration and number of instance models and the usage duration (in minutes). Billing rules:

- Billing by minute. Less than 1 minute is regarded as 1 minute.
- Billing starts when the order is submitted and the cluster instance is in the `Running` state.
- Release all resources and stop billing as soon as the cluster was deleted.
- When the cluster is stopped, the billing still continues because the resources have not been released. To stop billing, you must delete the cluster and release the resources.

Insufficient balance and arrears

- **Reminder of insufficient balance:**
  - Judge whether your account balance, including available vouchers, is sufficient to pay for the next 3 days accordance to your bill amount in the last 3 days; if insufficient, the system sends a renewal reminder.
  - Judge whether your account balance, including available vouchers, is sufficient to pay for the next day according to your bill amount for the last day; if insufficient, the system sends a renewal reminder.
- **Arrears handling:**
  - You are in arrears when your account balance is 0 yuan and you cannot pay the Palo service bill. The system sends you an arrears notice when you are in arrears.
  - The service is stopped immediately after you are in arrears, and then the system sends you an arrears notice for stopping service. In order not to affect your service, it is recommended that the amount in your account is sufficient to pay the bill when you are using Palo’s service.
  - After 7 days of being in arrears, the system deletes the service and release the resources if it has not been recharged.

Billing formula

- Fees = Unit price × Number of nodes × Service duration

Price

- **Balance type 1**

Node Configuration	CPU（Core）	RAM (GB)	SSD Cloud Disk Storage (GB)	Price (CNY/min/node)
Leader Node	4	16	100	0.0345
Compute Node	4	16	200	0.0370

- **Balance type 2**

Node Configuration	CPU（Core）	RAM (GB)	SSD Cloud Disk Storage (GB)	Price (CNY/min/node)
Leader Node	8	32	200	0.0691
Compute Node	8	32	500	0.0764

- **Balance type 3**

Node Configuration	CPU（Core）	RAM (GB)	SSD Cloud Disk Storage (GB)	Price (CNY/min/node)
Leader Node	16	32	200	0.0901
Compute Node	16	64	200	0.1332

- **Memory Type 1**

Node Configuration	CPU（Core）	RAM (GB)	SSD Cloud Disk Storage (GB)	Price (CNY/min/node)
Leader Node	8	32	200	0.0691
Compute Node	8	32	1024	0.0893

- **Memory Type 2**

Node Configuration	CPU（Core）	RAM (GB)	SSD Cloud Disk Storage (GB)	Price (CNY/min/node)
Leader Node	16	32	200	0.0901
Compute Node	32	64	4096	0.2707

- **Memory type 3**

Node Configuration	CPU (Core)	RAM (GB)	High-Performance Cloud Disk Storage (GB)	Price (CNY/min/node)
Leader Node	16	32	200	0.0901
Compute Node	16	64	4096	0.1635

- **Memory type 4**

Node Configuration	CPU (Core)	RAM (GB)	High-Performance Cloud Disk Storage (GB)	Price (CNY/min/node)
Leader Node	16	32	200	0.0901
Compute Node	16	64	2048	0.1460

- **Computing type 1**

Node Configuration	CPU（Core）	RAM (GB)	SSD Cloud Disk Storage (GB)	Price (CNY/min/node)
Leader Node	16	32	200	0.0901
Compute Node	32	64	500	0.1826

Prepaid

In case of prepayment mode, you can prepay the bill on a monthly basis. Palo calculates the monthly subscription price according to the configuration and number of the instance models you select. In general, the prepaid price for the same service duration is far lower than the on-demand billing price.

Expiration reminder

- Seven days before the Palo service expires, the system sends you a reminder of the impending expiration.
- Service is terminated immediately after expiration, and the system sends you an arrears notice. After stopping service for 7 days, if the service has not been renewed, the system deletes the service and release the resources.

Billing formula

- Fees = Unit price × Number of nodes × Service duration

Price

- Balance type 1

Node Configuration	CPU (Core)	RAM (GB)	SSD Cloud Disk Storage (GB)	Price (CNY/month/node)
Leader Node	4	16	100	700
Compute Node	4	16	200	800

- Balance type 2

Node Configuration	CPU (Core)	RAM (GB)	SSD Cloud Disk Storage (GB)	Price (CNY/month/node)
Leader Node	8	32	200	1400
Compute Node	8	32	500	1700

- Balance type 3

Node Configuration	CPU (Core)	RAM (GB)	SSD Cloud Disk Storage (GB)	Price (CNY/month/node)
Leader Node	16	32	200	1688
Compute Node	16	64	200	2428

- Memory type 1

Node Configuration	CPU (Core)	RAM (GB)	SSD Cloud Disk Storage (GB)	Price (CNY/month/node)
Leader Node	8	32	200	1319
Compute Node	8	32	1024	2184

- Memory type 2

Node Configuration	CPU (Core)	RAM (GB)	SSD Cloud Disk Storage (GB)	Price (CNY/month/node)
Leader Node	16	32	200	1688
Compute Node	32	64	4096	7258

- Memory type 3

Node Configuration	Node Configuration	RAM (GB)	High-Performance Cloud Disk Storage (GB)	Price (CNY/month/node)
Leader Node	16	32	200	1688
Compute Node	16	64	4096	3723

- **Memory type 4**

Node Configuration	Node Configuration	RAM (GB)	High-Performance Cloud Disk Storage (GB)	Price (CNY/month/node)
Leader Node	16	32	200	1688
Compute Node	16	64	2048	2970

- **Computing type 1**

Node Configuration	CPU (Core)	RAM (GB)	SSD Cloud Disk Storage (GB)	Price (CNY/month/node)
Leader Node	16	32	200	1688
Compute Node	32	64	500	3482

> Prepayment discount: 17% discount for one year, 30% discount for two years and 50% discount for three years.

Operation Guide

Create a Cluster

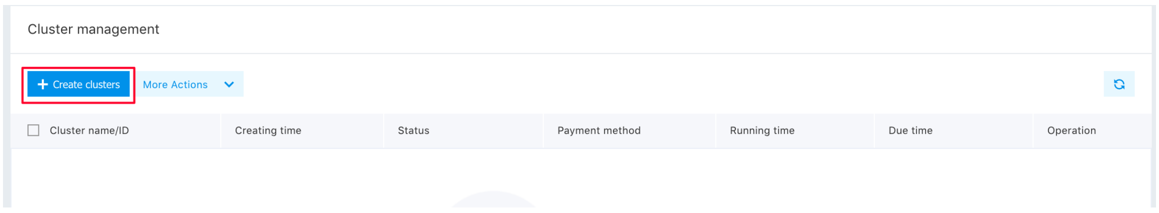
If you do not have a Baidu Open Cloud account, please register a Baidu account first(click Here to get started), log in to Baidu Open Cloud with your account and enter the Palo's main page.

Click Shop Now after entering the page.

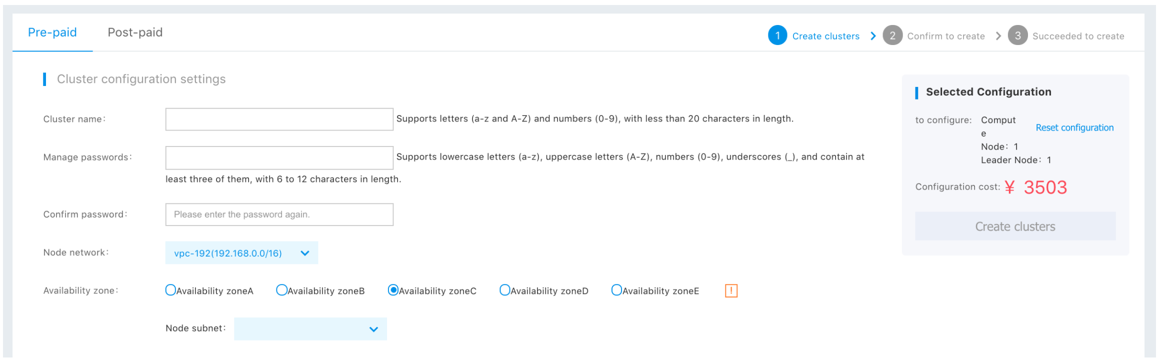
If you are creating a Palo cluster for the first time, the following interface pops up when entering the Palo cluster list page.

Please read the prompt popped up on the page and click open.

Then, you can enter the cluster list page. Click create cluster to enter the cluster configuration page:



Cluster name/ID	Creating time	Status	Payment method	Running time	Due time	Operation



Cluster name: Supports letters (a-z and A-Z) and numbers (0-9), with less than 20 characters in length.

Manage passwords: Supports lowercase letters (a-z), uppercase letters (A-Z), numbers (0-9), underscores (_), and contain at least three of them, with 6 to 12 characters in length.

Confirm password: Please enter the password again.

Node network: vpc-192(192.168.0.0/16)

Availability zone: Availability zoneA Availability zoneB [✓ Availability zoneC] Availability zoneD Availability zoneE

Node subnet:

Selected Configuration

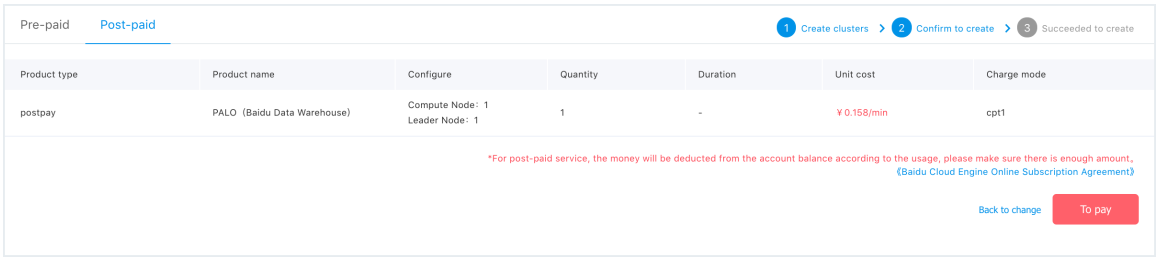
to configure: Compute Node: 1 Leader Node: 1

Reset configuration

Configuration cost: ¥ 3503

Create clusters

After selecting the configuration, click create cluster to enter the payment page:


Pre-paid Post-paid

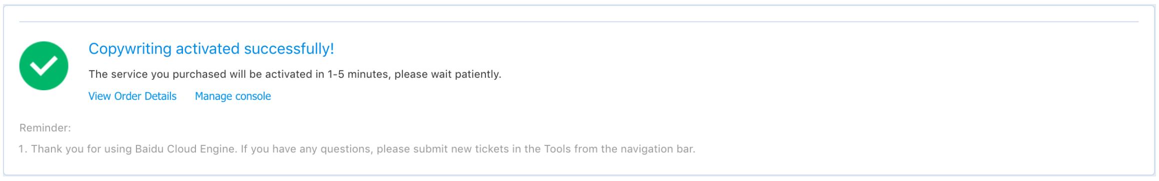
1 Create clusters > 2 Confirm to create > 3 Succeeded to create

Product type	Product name	Configure	Quantity	Duration	Unit cost	Charge mode
postpay	PALO（Baidu Data Warehouse）	Compute Node：1 Leader Node：1	1	-	￥0.158/min	cpt1

*For post-paid service, the money will be deducted from the account balance according to the usage, please make sure there is enough amount。
《Baidu Cloud Engine Online Subscription Agreement》

Back to change To pay

Click To pay to enter the payment page. After the payment is completed, the Opened successfully page appears:


Copywriting activated successfully!

The service you purchased will be activated in 1-5 minutes, please wait patiently.

View Order Details Manage console

Reminder:

1. Thank you for using Baidu Cloud Engine. If you have any questions, please submit new tickets in the Tools from the navigation bar.

Within about 1-5 minutes, Palo's cluster is created. You can log in to the console to view the clusters that is being created or has been created.

Perhaps, you cannot see the cluster in the console within a period after the order is created successfully. Please wait patiently for a few minutes.

Cluster Scaling

You can modify the size of the cluster at any time as needed after the cluster is created. On the page displaying the cluster details, click Cluster Scaling to enter the cluster scaling page, which displays the configuration and billing information of the current user cluster.

You can modify the desired number of clusters. At present, Palo only supports scaling nodes, and the feature of scaling nodes is under development. Among them, the Leader Node is allowed to be scaled to 3 nodes and Compute Node to up to 50 nodes.

If you are a prepaid user, the configuration fee is for the new configuration. If you are on-demand billing users, the configuration fee is the total unit price after being scaled.

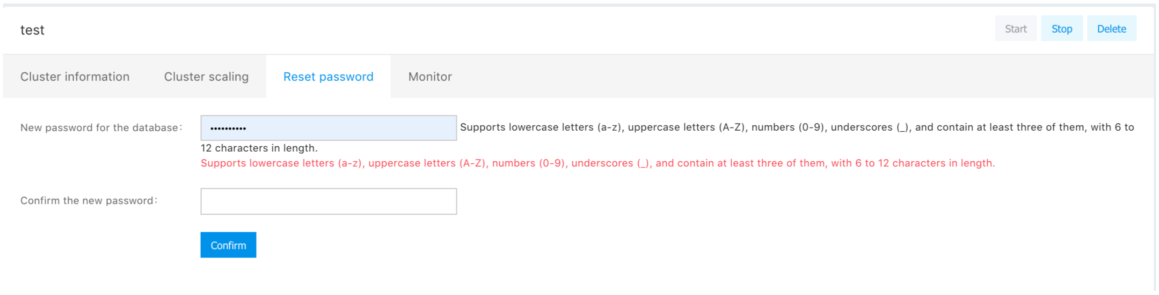
After the operation is done, click Cluster Scaling and confirm the order to finish the modification of cluster size.

At this moment, the cluster enters the Expanding state.

Meanwhile, it can be seen that scaling nodes are being created. The scaling can be completed in about 1-5 minutes. The status of the scaling node is changed to Running.

Reset the Password

When you forget the password of the admin, you can log in to the cluster management page, click Reset Password to enter the password modification page, and then type in the new password and verification code to complete the reset.


test

Start Stop Delete

Cluster information Cluster scaling Reset password Monitor

New password for the database： •••••••••• Supports lowercase letters (a-z), uppercase letters (A-Z), numbers (0-9), underscores (_), and contain at least three of them, with 6 to 12 characters in length.

Supports lowercase letters (a-z), uppercase letters (A-Z), numbers (0-9), underscores (_), and contain at least three of them, with 6 to 12 characters in length.

Confirm the new password：

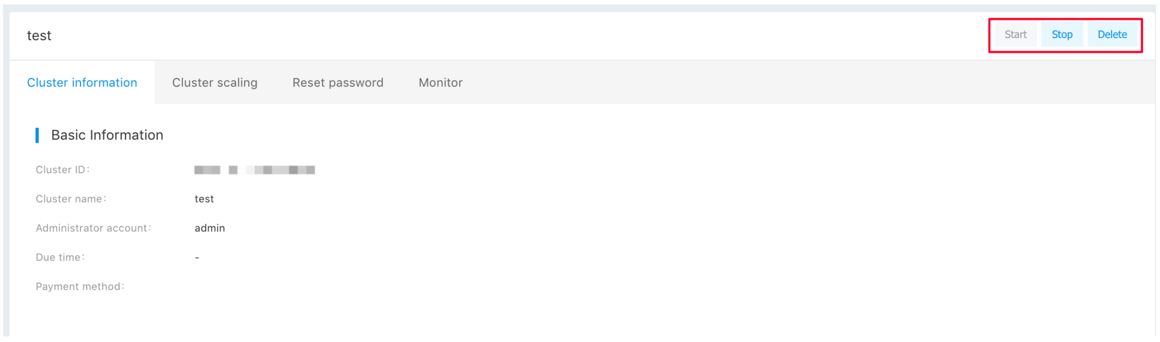
Confirm

Stop and Delete a Cluster

At present, Palo service provides the user the ability to stop and delete clusters. The difference between stop and delete is as follows:

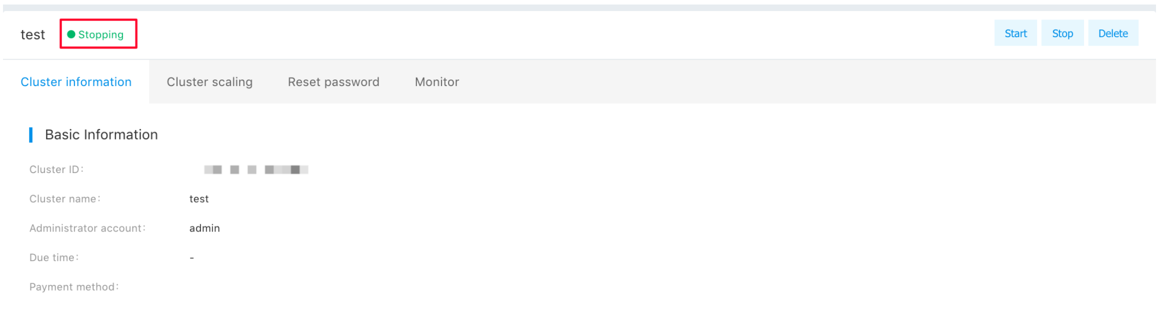
- Stop: Just stop the service, but do not release the resources occupied by the service, which is similar to stopping the local MySQL service. Therefore, it is still charged in the stop state. After the service is stopped, the service can be restarted through the start button. There is no need to re-import data after the service is started.
- Delete: Release the resources occupied by the service. After the data of the service is completely deleted, it cannot be restored, which is similar to deleting the local MySQL program and data.

The user can stop and delete the cluster by clicking the button `Stop` or `Delete` at the top right corner, as shown below:



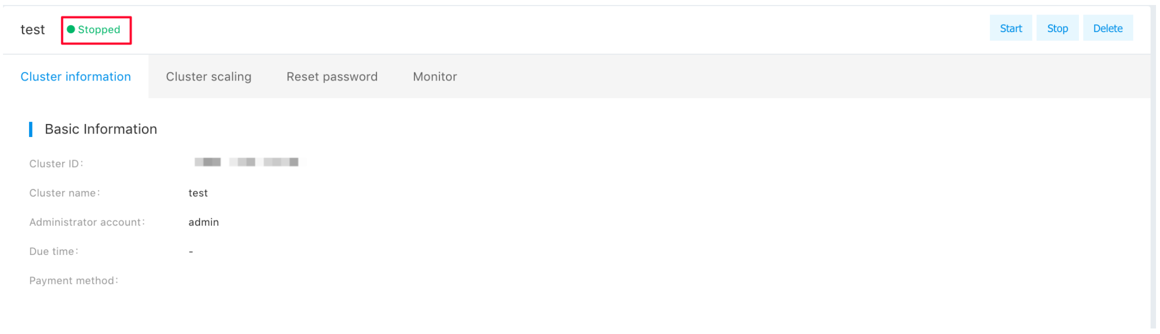
Cluster ID:	[redacted]
Cluster name:	test
Administrator account:	admin
Due time:	-
Payment method:	

When clicking the button `Stop`, the cluster enters a stopping state:



Cluster ID:	[redacted]
Cluster name:	test
Administrator account:	admin
Due time:	-
Payment method:	

After about 1 minute, the cluster is stopped:



Cluster ID:	[redacted]
Cluster name:	test
Administrator account:	admin
Due time:	-
Payment method:	

At this moment, the cluster is not accessible. Click the `Start` button to restart the cluster.

When clicking the `Delete` button, the cluster is deleted immediately. And, it is not visible from the console.

Monitoring and Alarm

This document mainly introduces Palo's monitoring items and related alarm configurations.

After entering the Palo cluster details page, the user can click the **Monitoring** tab to enter the monitoring page.

Palo displays different monitoring items based on different node types. The user can select **Leader Node** or **Compute Node** in the **Node type** option, select the instances the user wants to view in the **Monitoring object** (up to 10), and then select relevant monitoring items to view.

Introduction of Monitoring Metrics

Public Monitoring Metrics

Both Leader Node and Compute Node have the following common monitoring metrics:

1. CPU usage

   Display the CPU usage as a percentage. The higher the value, the higher the CPU load.

2. Memory usage

   Display the memory usage in GB.

3. Memory utilization

   Display the memory usage as a percentage. The higher the value, the greater the memory currently consumed by the node.

4. Disk space usage

   Display the disk space usage in GB for the node.

5. Disk utilization

   Display the disk utilization as a percentage for the node. Display of the usage rate for a single disk is not supported currently.

6. CPU Stolen

   Display the CPU stolen ratio as a percentage. This value is higher than 10%, which indicates that the CPU resources of this node may be severely occupied.

Unique Monitoring Index of Leader Node

1. Current number of connections

   Display the current number of connections to Leader Node through Mysql port.

2. Current metadata log id

   Display the up-to-date log id synchronized with the current Leader Node metadata. The specific data of this monitoring item is meaningless, thus you need to only check whether the up-to-date log id of each Leader Node is kept synchronous.

3. Queries per second

   The number of queries per second (QPS) is displayed. This data only counts the number of query requests connected and initiated through Mysql protocol.

4. Requests per second

   Display the number of requests per second (RPS). This data only counts the number of requests connected and initiated through Mysql protocol. The requests include all requests such as queries, DDL, DML, etc.

5. Number of query errors per second

   Display the number of error queries occurred per second. This data only counts the number of errors in query connected and initiated through Mysql protocol.

6. JVM Old area usage rate

   Display the usage rate of JVM Old area as a percentage. The higher the value, the higher the JVM memory usage rate.

7. JVM Young area usage rate

   Display the usage rate of JVM Young area as a percentage.

Unique Monitoring Index of Compute Node

1. Current number of threads

   Display the number of threads for the process in Compute Node.

2. Number of file handles

   Display the number of file handles opened by the process in Compute Node.

3. Write rate

   Display the write rate per second in bytes. It is the rate of actually writting the data into Palo by the import command.

4. Fetch rate

   Display the fetch rate per second in bytes. It is the reading rate generated during the query.

Introduction of Monitoring Metrics Legend

The above figure is an example showing the CPU usage of the Compute Node. Monitoring metrics of multiple Compute Nodes are displayed by lines in different color. Click the corresponding node in the legend to display the monitoring metric of a node separately. At the upper right corner of the monitoring page, you can also select the time period to be displayed.

Alarm Configuration

Currently, only alarm configuration is supported for monitoring metric of Leader Node. At present, separate alarm configuration is required for each Leader Node. (The alarm configuration of Compute Node and batch alarm configuration by node type is launched in the near future)

Click on the **alarm details** on the right side of the above figure to enter the alarm configuration page corresponding to Leader Node.

Green/red/yellow in the above figure indicate the status of each alarm item currently configured, respectively.

Click **Add policy** to start adding an Alarm Policy.

The above figure is an example. In the above figure, an alarm policy named`cpu_alert` is configured. This policy is set to alarm when the average CPU utilization rate within 10 minutes is greater than 70% for 3 times in succession. The alarm method is to notify the alarm object through short messages and mails. When the metric resumes normal, notify the alarm object by mail. If there is insufficient monitoring data, the alarm object is notified by mail.

After the alarm strategy is set, if an alarm is triggered, the corresponding notification is received and the historical alarm can be viewed in the **alarm event**.

The alarm needs about 5 minutes to take effect.

Leader Node Alarm Policy Practice

Here are the recommendations for alarm configuration of Leader Node.

1. CPU Utilization

   The CPU is utilized on the on-demand basis. For example, if the average utilization within 15 minutes is more than 90%, an alarm is given.

2. Disk space storage capacity usage rate

   It is recommended to alert when the disk space storage capacity usage rate is greater than 80%. At this moment, it may be necessary to clean up the data or expand the capacity.

3. Memory utilization

   The memory is used on the on-demand basis. For example, if the average utilization within 15 minutes is more than 80%, an alarm is given.

4. Current number of connections

   By default, Palo’s upper limit of connections for a single user to a single Leader Node is 100. Assuming there are 3 Palo users, if the number of connections to a single Leader Node exceeds 300, and the connection is refused. According to the number of users, reasonable alarm policies can be configured here.

5. Queries per second

   The queries per second is made on the on-demand basis. Set up reasonable alarm policies according to business usage.

6. Requests per second

   The queries per second is made on the on-demand basis. Set up reasonable alarm policies according to business usage.

7. Number of query errors per second

   It is set according to the tolerable quantity of businesses. For more radical cases, it can be set that if the sum within 5 minutes is greater than 0, an alarm should be given.

8. JVM Old area usage rate

   It is recommended to set the average usage rate of the Old area to be more than 75% within 15 minutes, and trigger an alarm after two consecutive times.

9. JVM Young area usage rate

   The usage rate of Young area has no actual alarm significance. Usually, just look at the trend.

Privilege and Sub-account

This document mainly describes the operation privileges of public cloud accounts and related sub-accounts on Palo cluster.

The user can set sub-account and related privileges in the **Multi-user Access** at the upper right corner after logging in to the public cloud.

Type of Privileges

Product-level Privileges

The product-level privileges are applied to all Palo cluster instances under the primary account.

1. Management privileges

   Policy name: PALOFullControlPolicy

   Note:

   PALO cluster administrative privileges

   This privilege can be used to perform the following operations:

   - Create, delete and expand instances;
   - View all instance information;
   - Start and stop instances;
   - Reset password for instance;
   - Bind and unbind EIP for instances;
   - Set up monitoring alarm;

2. Operations privileges

   Policy name: PALOOperatePolicy

   Note:

   PALO cluster Operations privilegeege

   This privilege can be used to perform the following operations:

   - View all instance information;

- Start and stop instances;
- Reset password for instance;
- Bind and unbind EIP for instances;
- Set up monitoring alarm;

3. Read-only privilege

   Policy name: PALOReadPolicy

   Note:

   PALO cluster read-nnly privilege

   This privilege can be used to perform the following operations:

   - View all instance information;

Instance-level Privileges

Instance-level privileges are applied to the specified Palo cluster instance. The user needs to create policies with corresponding privileges through policy management, and then can assign policies to sub-accounts.

1. Operations privileges

This privilege can be used to perform the following operations:

- Start and stop the specified instance;
- Reset the password for the specified instance;
- Bind and unbind EIP for the specified instances;
- Set up monitoring alarm;

2. Read-only privilege

This privilege can be used to perform the following operations:

- View the specified instance information;

Note:

The Operations privilegeege does not include **View the specified instance information**. Therefore, when configuring policies for OPS sub-accounts, it may be necessary to select both **Operations privilege** and **Read-only privilege**.

Privilege Description

The primary account can create sub-accounts and assign product-level or instance-level privileges to the sub-accounts. The final privileges that the sub-account has are the union of all assigned privileges.

For example, sub-account A is assigned product-level Operations privileges and read-only privileges of the instance X. Then, sub-account A also has the Operations privileges for instance X.

Privilege-Palo Privilege Management

Palo's new access control system refers to Mysql's access control mechanism, achieves granular access control at the table level, and supports the whitelist mechanism.

Interpretation of Terms

1. user_identity

In the privilege system, a user is identified as a User Identity. The user identity consists of two parts, i.e., username and userhost, of which the username is the user name and consists of uppercase and lowercase English letters, and the userhost represents the account IP that is linked. The user_identity, presented in the form of username@‘userhost', represents the username from the userhost.

Another representation of the user_identity is username@['domain'], where the domain means the domain name and can be resolved into a group of IPs through DNS and BNS (Baidu Name Service). Finally, it is represented by a group of username@'userhost'. Thus, it’s required to use username@'userhost' to represent it.

2. Privilege

   The object of privilege is node, database or table. Different privileges represent different operation permissions.

3. Role

   Palo can create a custom-named role. A role can be viewed as a set of permissions. The new user can be assigned a certain role, and thus automatically assigned the privileges that the role has. Subsequent privilege change to the role is also reflected in all user privileges belonging to the role.

4. user_property

   The user property is directly attached to a user, but not the user identity. That is to say, both cmy@'192.%' and cmy@['domain'] have the same group of user properties which belong to user cmy, instead of cmy@'192.%' or cmy@['domain'].

   The user property includes, but is not limited to: Maximum number of user connections, imported cluster configuration, etc.

Supported Operations

1. CREATE USER
2. DROP USER
3. GRANT
4. REVOKE
5. CREATE ROLE
6. DROP ROLE
7. SHOW GRANTS
8. SHOW ALL GRANTS
9. SHOW ROELS
10. SHOW PROPERTY

For detailed help information on the above commands, you can use help+command for help after connecting Palo by the mysql client, such as `help create user`.

Privilege Description

Palo currently supports the following privileges

1. Node_priv

Node change privilege It includes addition, deletion, downlining and other operations of FE, BE, and BROKER nodes. At present, such privilege can only be granted to Root users.

2. Grant_priv

Permission change privilege It is allowed to perform such operations as authorization, revocation, addition/deletion/change of users/roles, etc.

3. Select_priv

Read-only privileges on databases and tables.

4. Load_priv

Write privileges on databases and tables. It includes privileges to Load, Insert, Delete, etc.

5. Alter_priv

Modification privileges on databases and tables. It includes: renaming libraries/tables, adding/deleting/changing columns, etc.

6. Create_priv

Privileges to create databases and tables.

7. Drop_priv

Privilege to delete databases and tables.

Other Explanations

1. When Palo is being initialized, the following users and roles are automatically created:
    1. Operator role: This role has Node_priv and Admin_priv, i.e., all privileges to Palo. In a subsequent upgrade version, the privileges of this role are limited to Node_priv (i.e., only the node change privileges are granted) to meet deployment requirement on some clouds.
    2. Admin role: This role has Admin_priv, i.e., all privileges other than node change privilege.
    3. Root @'%': It means a root user, who is allowed to log in from any node, and whose role is operator.
    4. Admin @'%': It means an Admin user, who is allowed to log in from any node, and whose role is admin.
2. Deletion of or change in the privileges of a role or user created by default is not supported.
3. There is only one user in the operator role. Multiple users in the admin role can be created.
4. Operation instructions that may result in conflicts
    1. Conflict of the domain name with IP:

       Supposing the following user is created:

       CREATE USER cmy@['domain'];

       And give authorization:

       GRANT SELECT_PRIV ON *.* TO cmy@['domain']

       The domain is resolved into two IPs, i.e., IP1 and IP2.

       After assumption, cmy@'ip1' is authorized separately:

       GRANT ALTER_PRIV ON *.* TO cmy@'ip1';

       Then, the privileges of cmy@'ip1' are modified to SELECT_PRIV, ALTER_PRIV. And, when the privileges of cmy@['domain'] is changed again, cmy@'ip1' is not changed accordingly.
    2. Repeated IP conflict:

       Supposing the following user is created:

CREATE USER cmy@'%' IDENTIFIED BY "12345";

CREATE USER cmy@'192.%' IDENTIFIED BY "abcde";

On the priority level, '192.%' takes priority over '%'; therefore, when the user cmy attempts to log into Palo using password' 12345' from 192.168.1.1, the user is refused.

3. Privilege conflict

5. Forget password

   If you forget your password and cannot log in to Palo, you can log in to Palo without a password using the following command on the machine where the Palo FE node is located:

   `mysql-client -h 127.0.0.1 -P query_port -uroot`

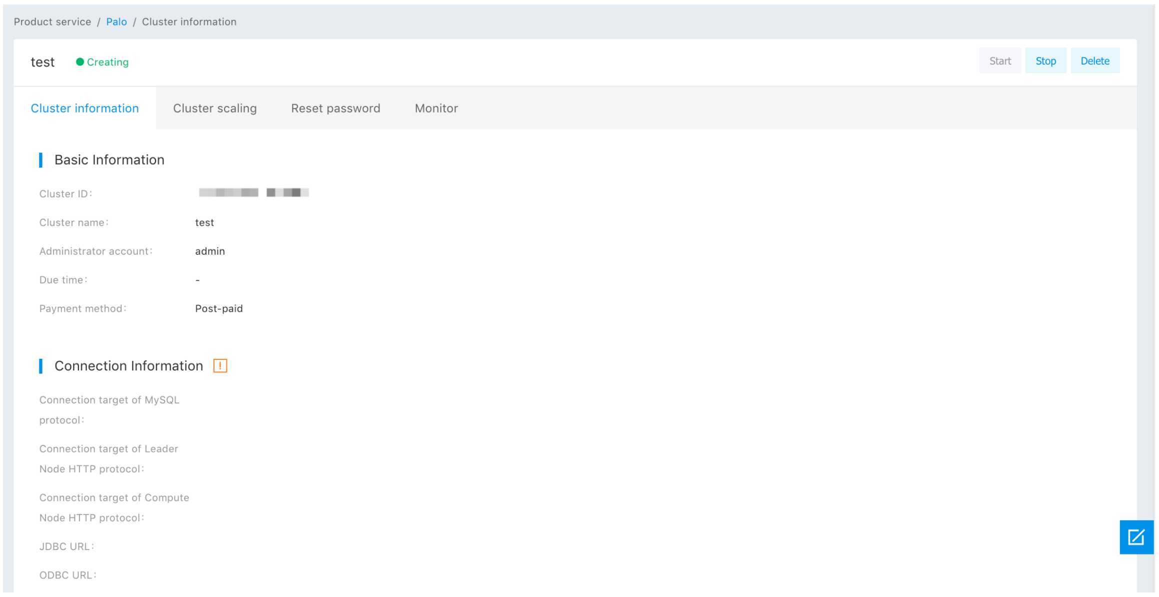
   After log-in, the password can be reset by the SET PASSWORD command.

Getting Started

Connection

Palo provides two access ports, i.e., 9030 port supporting Mysql protocol and 8030 port supporting http protocol.

After creating the Palo cluster, you can view the connection information of the cluster on the `Cluster Information` page.


Product service / Palo / Cluster information

test ● Creating

Start Stop Delete

Cluster information | Cluster scaling | Reset password | Monitor

Basic Information

Cluster ID: [redacted]

Cluster name: test

Administrator account: admin

Due time: -

Payment method: Post-paid

Connection Information

Connection target of MySQL protocol:

Connection target of Leader Node HTTP protocol:

Connection target of Compute Node HTTP protocol:

JDBC URL:

ODBC URL:

Notes:

1. Cluster Name: Means the name customized when the Palo cluster is created.
2. Administrator’s account: Means the name of the account with Palo administrator privileges, which is admin by default.
3. Expiration time: If you are a prepaid user, the expiration time of your order is displayed here. However, if you are an on-demand billing user, no information is displayed here.
4. MySQL Protocol Connection target: Use this target when you connect to Palo using the MySQL protocol.
5. Leader Node HTTP Protocol Connection Target: This port provides a simple Palo WebUI. Meanwhile, you can also import local files through this target.
6. Compute Node HTTP protocol connection target: This target can be used for importing local files.
7. JDBC/ODBC URL: They are examples for catenated string of MySQL JDBC/ODBC, respectively. Among them, DB_NAME, USER_NAME and PASSWORD should be replaced with corresponding true values.

Connection by Mysql Protocol

You can use the following command to connect Palo through mysql-client (As an example, mysql-5.7.22 client is used here):

`./mysql -h 192.168.16.17 -P 9030 -uadmin -pyour_password`

Http Protocol Connection

Currently, Palo's Http Server mainly provides the following two features:

1. WebUI

   Palo provides a simple WebUI to view some internal operating status of the Palo system. You can directly enter the HTTP protocol connection target in the browser within the same VPC, and then enter the administrator account and password to log in to WebUI.

2. Local file import

   Palo provides a way to import local files. This import method realizes data upload through http protocol. The specific usage is demonstrated in the *Importing data* section.

Create a Database

After connecting to Palo, you can start creating your first Palo database by executing SQL statements:

`CREATE DATABASE testDb;`

Create a Table

Now, a testTable can be created in the testDb database. The tabling statement is:

```
 CREATE TABLE testDb.testTable
(
  k1 bigint,
  k2 varchar(100),
  v varchar(100) REPLACE
) DISTRIBUTED BY HASH(k1) BUCKETS 8;
```

This statement is executed to create a testTable that contains 3 columns. Among them, k1 and k2 are key columns, and v is value column, and the aggregation method for v is REPLACE. Furthermore, the table is bucketed in Hash algorithm based on the k1 value, and the number of buckets is 8.

After the table is created successfully, you can view the created table through `show tables;` statements.

For more instructions for table creating, you can click `help create table;`.

Note: Only one replica table can be created if you have only one Compute Node. Here’s the statement:

```
 CREATE TABLE testDb.testTable
(
    k1 bigint,
    k2 varchar(100),
    v varchar(100) REPLACE
) DISTRIBUTED BY HASH(k1) BUCKETS 8
PROPERTIES
(
    "replication_num" = "1"
);
```

Import the Data

Palo now supports two data import methods, i.e., BOS file import and local file import.

1. BOS file import

   BOS is short for Baidu Object Storage. For information on how to enable and use BOS, click Here.

   Palo reads the data on BOS through a process named broker and imports it into the Palo storage engine.

   Assuming that you already have a data file on BOS:

   `bos://yourt_buckets/path/to/file.txt`

   The content is two lines of data separated by commas:

   ```
    10000, Smith, Hello Palo!
   10001, Jack, Let's loading.
   ```

   Its corresponding table structure is the `testTable` created in the previous section.

   You can import data through the following command:

   ```
    LOAD LABEL testDb.first_label (
      DATA INFILE("bos://your_buckets/path/to/file.txt")
      INTO TABLE testTable
      COLUMNS TERMINATED BY ","
   )
   WITH BROKER bos (
       "bos_endpoint" = "http://bj.bcebos.com",
       "bos_accesskey" = "xxxxxxxxxxxxxxxxxxxx",
       "bos_secret_accesskey"="xxxxxxxxxxxxxxxxxxxx"
   );
   ```

   The `first_label` is the label to finish this import, which is specified by the user. For each import, a new label must be specified to differentiate the imported batches. `bos` is the default name of Broker. You can view it through `show broker;`. The following parameters are required for access to the bos.

   Import is an asynchronous command. When the above import command is executed successfully, it only means that the import task is submitted successfully. You need to view the execution of the task through `show load;`. When the `State` in the result is `FINIHSED`, it means that the import is done successfully, and the data is visible.

   Access to BOS in non-PALO area is not supported. For example, the PALO cluster in Beijing cannot access the BOS in Baoding.

   `bos_endpoint` specifies the endpoint of BOS. For example, BOS in Beijing is abbreviated to bj while Baoding is bd.

   `bos_accesskey` and `bos_secret_accesskey` are AK and SK for accessing BOS. They can be viewed in `Security Certification` at the upper right corner of the public cloud page.

   You can access `help broker load;` and `help show load;` to find more help information on broker import.

2. Importing local file

Palo also supports the direct import of local files. `Local` refers to the server in the same VPC as PALO.

Local file import is the only feature that Palo needs to use the HTTP protocol port to operate.

`Leader Node HTTP Protocol Connection Target` and `Compute Node HTTP Protocol Connection Target` can be found on the cluster page.

In theory, you can import local files by using any HTTP connection target. However, it’s suggested that you use `Compute Node HTTP Protocol Connection Target` directly to reduce redirect once. And, it is required to avoid the problem that the client is not connected with the Compute Node.

You can import local files through `curl`.

`curl --location-trusted -u user:password -T data.txt http://host:port/api/testDb/testTbl/_load?label=second_label&column_separator=%2c`

Of which, data. txt is the local file you want to import. TestDb and testTbl are target library tables that need to be imported. And, the label is the one that is imported this time, and the column_separator specifies the column separator as "comma" (the url code is %2c) for the imported label,

If the submission gets done successfully, it returns the following content:

```
  {
    "status": "Success",
    "msg": "OK"
}
```

Similarly, you can view the status of import tasks through http commands.

`curl --location-trusted -u user:password http://192.168.16.17:8030/api/testDb/_load_info?label=second_label`

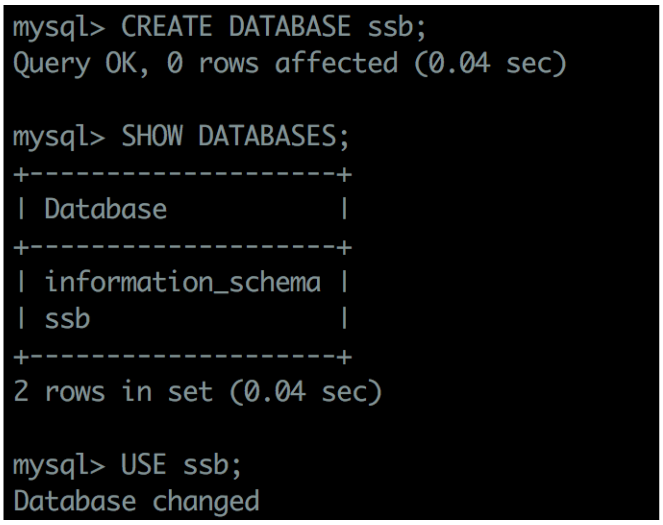
More help information on this import method can be viewed through `help mini load`.

Query

Palo supports access to it by Mysql protocol. You can access Palo by using any client or program library using only standard Mysql.

Query by MySQL

You can view the database through `show databases;` after connecting to Palo through MySQL client, Use `use db_name;` to select a database, and `show tables;` to view tables in the database.



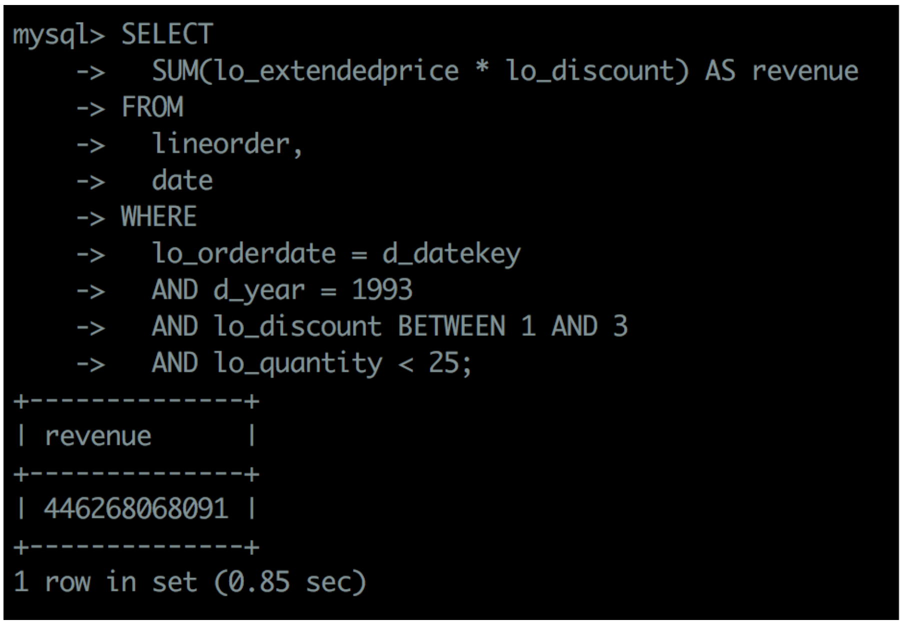
```
mysql> CREATE DATABASE ssb;
Query OK, 0 rows affected (0.04 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| ssb                |
+--------------------+
2 rows in set (0.04 sec)

mysql> USE ssb;
Database changed
```

You can ask for help to create databases and tables through `help create database;` and `help create table;` .

When executing a query, simply enter the SQL statement directly



```
mysql> SELECT
    ->   SUM(lo_extendedprice * lo_discount) AS revenue
    -> FROM
    ->   lineorder,
    ->   date
    -> WHERE
    ->   lo_orderdate = d_datekey
    ->   AND d_year = 1993
    ->   AND lo_discount BETWEEN 1 AND 3
    ->   AND lo_quantity < 25;
+--------------+
| revenue      |
+--------------+
| 446268068091 |
+--------------+
1 row in set (0.85 sec)
```

Query by JDBC/ODBC

You can connect to Palo by using JDBC/ODBC whose connection url is available on the cluster information page. The following gives a JDBC sample code, which is driven by JDBC of the MySQL.

```
  import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;

public class Test {
    public static void main(String[] args) {
        // Load the driver
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }

        // Create a database connection
        java.sql.Connection conn = null;
        try {
            String url = "jdbc:mysql://your_cluster_node:9030/testDb";
            String user = "admin";
            String password = "my_password";
            conn = DriverManager.getConnection(url, user, password);
        } catch (SQLException e) {
          e.printStackTrace();
        }

        // Initiate a query and get the results
        java.sql.Statement stmt = null;
        ResultSet rs = null;
        try {
            stmt = conn.createStatement();
            rs = stmt.executeQuery("select * from testTable limit 10");
            while (rs.next()) {
                System.out.print(rs.getInt("k1") + " ");
                System.out.println(rs.getString("k2") + " ");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        // Close the database connection
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

SQL Manual

Data Type

TINYINT Data Type

Length: Signed integer with length of 1 byte.

Range: [-128, 127]

Conversion: Palo can automatically convert this type to a larger integer or floating point type. Use the CAST () function to convert it to CHAR.

Example:

```
  mysql> select cast(100 as char);
+------------------------+
| CAST(100 AS CHARACTER) |
+------------------------+
| 100                    |
+------------------------+
1 row in set (0.00 sec)
```

SMALLINT Data Type

Length: Signed integer with length of 2 byte.

Range: [-32768, 32767]

Conversion: Palo can automatically convert this type to a larger integer or floating point type. Use the CAST () function to convert it to TINYINT, CHAR.

Example:

```
  mysql> select cast(10000 as char);
+--------------------------+
| CAST(10000 AS CHARACTER) |
+--------------------------+
| 10000                    |
+--------------------------+
1 row in set (0.01 sec)

mysql> select cast(10000 as tinyint);
+------------------------+
| CAST(10000 AS TINYINT) |
+------------------------+
|                     16 |
+------------------------+
1 row in set (0.00 sec)
```

INT Data Type

Length: Signed integer with length of 4 byte.

Range: [-2147483648, 2147483647]

Conversion: Palo can automatically convert this type to a larger integer or floating point type. Use the CAST () function to convert it to TINYINT, SMALLINT, CHAR.

Example:

```
  mysql> select cast(111111111  as char);
+------------------------------+
| CAST(111111111 AS CHARACTER) |
+------------------------------+
| 111111111                    |
+------------------------------+
1 row in set (0.01 sec)
```

BIGINT Data Type

Length: Signed integer with length of 8 byte.

Range: [-9223372036854775808, 9223372036854775807]

Conversion: Palo can automatically convert this type to a larger integer or floating point type. Use the CAST () function to convert it to TINYINT, SMALLINT, INT, CHAR.

Example:

```
  mysql> select cast(9223372036854775807 as char);
+----------------------------------------+
| CAST(9223372036854775807 AS CHARACTER) |
+----------------------------------------+
| 9223372036854775807                    |
+----------------------------------------+
1 row in set (0.01 sec)
```

LARGEINT Data Type

Length: Signed integer with length of 16 byte.

Range: [-2^127, 2^127-1]

Conversion: Palo can automatically convert this type to a floating-point type. Use the CAST () function to convert it to TINYINT, SMALLINT, INT, BIGINT, CHAR

Example:

```
  mysql> select cast(922337203685477582342342 as double);
+------------------------------------------+
| CAST(922337203685477582342342 AS DOUBLE) |
+------------------------------------------+
|                    9.223372036854776e23 |
+------------------------------------------+
1 row in set (0.05 sec)
```

FLOAT Data Type

Length: Floating point type with length of 4 bytes.

Range: -3.40E+38 ~ +3.40E+38。

Conversion: Palo automatically converts FLOAT type to DOUBLE type. Users can use CAST () to convert it to TINYINT, SMALLINT, INT, BIGINT, STRING, TIMESTAMP.

DOUBLE Data Type

Length: Floating point type with a length of 8 bytes.

Range: -1.79E+308 ~ +1.79E+308。

Conversion: Palo does not automatically convert DOUBLE types to other types. Users can use CAST () to convert it to TINYINT, SMALLINT, INT, BIGINT, STRING, TIMESTAMP. Users can use index symbols to describe the DOUBLE type or obtain it through STRING conversion.

DECIMAL Data Type

DECIMAL[M, D]

Decimal type to ensure accuracy. M represents the total number of valid digits, and D represents the maximum number of digits after the decimal point. The range of m is [1,27], the range of d is [1,9]. In addition, M must be greater than or equal to the value of D. The default value is decimal[10,0].

precision: 1 ~ 27

scale: 0 ~ 9

Example:

1. The default value is decimal(10, 0)

   mysql> CREATE TABLE testTable1 (k1 bigint, k2 varchar(100), v decimal SUM) DISTRIBUTED BY RANDOM BUCKETS 8; Query OK, 0 rows affected (0.09 sec)

```
  mysql> describe testTable1;
+-------+----------------+------+-------+---------+-------+
| Field | Type           | Null | Key   | Default | Extra |
+-------+----------------+------+-------+---------+-------+
| k1    | bigint(20)     | Yes  | true  | N/A     |       |
| k2    | varchar(100)   | Yes  | true  | N/A     |       |
| v     | decimal(10, 0) | Yes  | false | N/A     | SUM   |
+-------+----------------+------+-------+---------+-------+
3 rows in set (0.01 sec)
```

2. Specify the value range of decimal explicitly

   CREATE TABLE testTable2 (k1 bigint, k2 varchar(100), v decimal(8,5) SUM) DISTRIBUTED BY RANDOM BUCKETS 8; Query OK, 0 rows affected (0.11 sec)

```
  mysql> describe testTable2;
+-------+---------------+------+-------+---------+-------+
| Field | Type          | Null | Key   | Default | Extra |
+-------+---------------+------+-------+---------+-------+
| k1    | bigint(20)    | Yes  | true  | N/A     |       |
| k2    | varchar(100)  | Yes  | true  | N/A     |       |
| v     | decimal(8, 5) | Yes  | false | N/A     | SUM   |
+-------+---------------+------+-------+---------+-------+
3 rows in set (0.00 sec)
```

DATE Data Type

Range: ['1000-01-01', '9999-12-31']. The default printing format is' YYYY-MM-DD'.

DATETIME Data Type

Range: ['1000-01-01 00:00:00', '9999-12-31 00:00:00']. The default print format is' YYYY-MM-DD HH:MM:SS'.

CHAR Data Type

Range: Char[(length)], a fixed-length string, the length of which ranges from 1 to 255 and is defaulted as 1.

Conversion: users can convert CHAR type to TINYINT, SMALLINT, INT, BIGINT, LARGEINT, DOUBLE, DATE or DATETIME type through CAST function.

Example:

```
  mysql> select cast(1234 as bigint);
+----------------------+
| CAST(1234 AS BIGINT) |
+----------------------+
|                 1234 |
+----------------------+
1 row in set (0.01 sec)
```

VARCHAR Data Type

Range: Char(length), variable length string, length range of 1~65535.

Conversion: users can convert CHAR type to TINYINT, SMALLINT, INT, BIGINT, LARGEINT, DOUBLE, DATE or DATETIME type through CAST function.

Example:

```
 mysql> select cast('2011-01-01' as date);
+----------------------------+
| CAST('2011-01-01' AS DATE) |
+----------------------------+
| 2011-01-01                 |
+----------------------------+
1 row in set (0.01 sec)

mysql> select cast('2011-01-01' as datetime);
+--------------------------------+
| CAST('2011-01-01' AS DATETIME) |
+--------------------------------+
| 2011-01-01 00:00:00            |
+--------------------------------+
1 row in set (0.01 sec)

mysql> select cast(3423 as bigint);
+----------------------+
| CAST(3423 AS BIGINT) |
+----------------------+
|                 3423 |
+----------------------+
1 row in set (0.01 sec)
```

HLL Data Type

Range: char(length); Length range: 1-16385. The user does not need to specify the length and default value, the length is controlled in the system according to the aggregation degree of data, and the HLL column can only be queried or used through the matching hll_union_agg, hll_cardinality and hll_hash.

Literal Constant

Each data type in Palo corresponds to a Literal of such type. The user can specify Literal in SQL statements, such as in the LIST of SELECT , in the WHERE clauses, and in the function’s parameters.

Numeric Literal Constant

The literal constants of integer types, such as TINYINT, SMALLINT, INT, and BIGINT, are a series of numbers that can be preceded by 0.

The literal constant of floating-point type, such as DOUBLE, is a series of numbers, optionally with decimal dot representing characters.

When required, the integer types can be promoted to floating-point type depending on the context.

When describing literal constants, exponential symbols, such as character e, can be used . For example, 1e+6 represents 10 to the sixth power, i.e., 1 million. Literal constants containing an exponent sign are recognized as floating point type.

String Literal Constant

String literal constants are put in single or double quotation marks. The literal constant of the string also includes the ones in other forms: the literal constant of the string is a string containing single quotation marks, which are enclosed within double quotation marks; the literal constant of a string is a string containing double quotation marks, which is put within single quotation marks.

To describe special characters of literal constant of a string, escape characters (character \ ) need to be added before the special characters.

- \t means the tab key
- \n means the newline character
- \r means the enter character
- \b means the fallback character
- \0 means the null characters of ASCII code, which is different from the NULL in SQL
- \Z means the end-of-text character in dos environment.
- \% and _ are used to escape wildcard characters in strings passed to the operator LIKE
- \ \ prevents backslash from being interpreted as escape characters
- If the literal constant of a string is enclosed in single or double quotation marks, the backslash can be used to escape the single or double quotation marks that appear in the literal constant of the string.
- If the character appearing after \ is not the escape character listed above, the character should remain unchanged and should not be escaped.

Date Literal Constant

Palo automatically converts CHAR type literal constants to DATE type literal constants. The time type literal constant accepted by Palo is in the input format of YYYY-MM-DD HH: MM: SS. ssssss, or contains the date only. The number (milliseconds) after decimal point in the above-mentioned format may be given or ignored. For example, the user can specify the time type as '2010-01-01', or '2010-01-01 10:10:10'.

SQL Operator

SQL operators are a series of functions used for comparison, which are widely used in WHERE clauses of SELECT statement.

Arithmetic Operators

Arithmetic operators usually appear in expressions consisting of left operands, operators, and (in most cases) right operands.

- + and - can be used as unary or binary operator. When it is used as a unary operator, such as +1, -2.5 or -col_name, it means that the value is multiplied by +1 or -1. Therefore, the unary operator + returns the unchanged value, while the unary operator - changes the symbolic bit of the value. The user can superimpose two unary operators, such as ++5 which returns a positive value, -+2 or +-2 which returns a negative value. However, the user cannot use -- because -- is interpreted as the remark statement. However, -- can be used only when a space or parenthesis is inserted between them, such as -(-2) or - -2, which means that the actual expression result is +2. When + or - is used as a binary operator such as 2+2, 3+1.5 or col1 + col2, the expression means adding or subtracting the right value from the left value. The left and right values must be numeric type.
- * and / represent multiplication and division, respectively. Operands on both sides of the operator must be data type. When two numerals are multiplied, operands of smaller type may be promoted, e.g., SMALLINT is promoted to INT or BIGINT, etc. If necessary, the result of expression is promoted to the next larger type, e.g., the type of result generated by TINYINT multiplied by INT is BIGINT. When two numerals are multiplied, the operand and the expression result are both interpreted as DOUBLE type to avoid loss of precision. To convert the expression result to another type, the user needs to convert it

with CAST function.

- %: Modulo operator. Returns the remainder of the left operand divided by the right operand. Both the left and right operands must be integer type.
- &, | and ^: The bitwise operator. It returns the result for the Bitwise_and, Bitwise_or, or Bitwise_Xor operation of two operands. Both the two operands are of integrator type. If the two operands of the bitwise operator are a different type, the smaller operand gets increased to a bigger operand for bitwise operation. One expression can have several arithmetic operators. The user can use parentheses to enclose the corresponding arithmetic expression. The arithmetic operand often has not an appropriate mathematical function to express the same feature as the arithmetic operator, e.g., no MOD() function indicates the feature of the % operand. In contrast, the mathematical function has not an appropriate arithmetic operand, e.g., the power function POW() does not have an appropriate ** power operand. Thus, the user can read the Mathematical Function section to know which arithmetic functions are supported.

“Between” Operator

In WHERE clauses, expressions may be compared with upper and lower bounds at the same time. If the expression is greater than or equal to the lower bound and less than or equal to the upper bound, the comparison result is TRUE.

Syntax:

```
expression BETWEEN lower_bound AND upper_bound
```

Data type: Usually, the calculation result of expression is numeric type, and this operator also supports other data types. If you must ensure that the lower and upper bounds are comparable characters, you can use the cast () function.

Instructions for use: Be careful when using string operands. Long strings starting with the upper bound does not match the upper bound, which is larger than the upper bound. Between 'A' and 'M' cannot match 'mJ'. To ensure that the expression works, some functions can be used, such as upper(), lower(), substr(), and trim ().

For example:

```
mysql> select c1 from t1 where month between 1 and 6;
```

Comparison Operators

The comparison operator is used to determine whether a column is equal to other or to sort columns. =, !=, <>, <, <=, >, and >= can be applied for all data types. Among them, <> means not equal to, as same as !=. Operators IN and BETWEEN provide shorter expressions to describe the comparison of such relationships as equal, less than, size, etc.

Operator IN

Operator IN is compared with the VALUE set and returns TRUE if it can match any element in the set. The parameter and VALUE set must be comparable. All expressions using the operator IN can be written as equivalent comparison connected by OR. But the syntax of the operator IN is simpler, more accurate, and easier than OR for Palo to optimize.

Example:

```
mysql> select * from small_table where tiny_column in (1,2);
```

Operator LIKE

This operator is used to compare strings. _ is used to match a single character, and % is used to match multiple characters. Parameters must match a complete string. In general, putting % at the end of a string is more practical.

Example:

```
 mysql> select varchar_column from small_table where varchar_column like 'm%';
+----------------+
| varchar_column |
+----------------+
| milan          |
+----------------+
1 row in set (0.02 sec)

mysql> select varchar_column from small_table where varchar_column like 'm____';
+----------------+
| varchar_column |
+----------------+
| milan          |
+----------------+
1 row in set (0.01 sec)
```

Logical Operators

The logical operator returns a BOOL value. The logical operator includes unary operator and plurality operator. The parameters processed by each operator are expressions that return a BOOL value. Supported operators are:

- AND: This is a binary operator, which returns TRUE if the calculation result of the left and right parameters are TRUE.
- OR: This binary operator returns TRUE if the calculation result for either of the left and right parameters is TRUE. If both parameters are FALSE, the OR operator returns FALSE.
- NOT: This is a unary operator that inverts the result of the expression. If the parameter is TRUE, the operator returns FALSE. If the parameter is FALSE, the operator returns TRUE.

Example:

```
 mysql> select true and true;
+-------------------+
| (TRUE) AND (TRUE) |
+-------------------+
|                 1 |
+-------------------+
1 row in set (0.00 sec)

mysql> select true and false;
+--------------------+
| (TRUE) AND (FALSE) |
+--------------------+
|                  0 |
+--------------------+
1 row in set (0.01 sec)

mysql> select true or false;
+-------------------+
| (TRUE) OR (FALSE) |
+-------------------+
|                 1 |
+-------------------+
1 row in set (0.01 sec)

mysql> select not true;
+----------+
| NOT TRUE |
+----------+
|        0 |
+----------+
1 row in set (0.01 sec)
```

Regular Expression Operators

Judge whether the parameters match regular expressions. Use regular expressions meeting POSIX standard to match the header of a string, $ to match the tail of a string, . to match any single character, * to match 0 or more options, + to match 1 or more options, ? express points greedy expressions, etc.. Regular expressions need to match the entire value, not just part of the string. If you want to match the middle part, the front part of the regular expression can be written as . * or . *. In general, ^ and $ may be omitted. Operators RLKIE and REGEXP are synonymous. |The operator is optional. |The regular expressions on both sides can only meet the conditions for one side, |and you often need to enclose the operators and the regular expressions on both sides with ().

Example:

```
  mysql> select varchar_column from small_table where varchar_column regexp '(mi|MI).*';
+----------------+
| varchar_column |
+----------------+
| milan          |
+----------------+
1 row in set (0.01 sec)

mysql> select varchar_column from small_table where varchar_column regexp 'm.*';
+----------------+
| varchar_column |
+----------------+
| milan          |
+----------------+
1 row in set (0.01 sec)
```

Alias

When you write the name of tables, columns, or expressions that contain columns in a query, you can assign them an alias at the same time. Whenv using table and column names, you can use their aliases to access them. Aliases are usually shorter and easier to remember than their original names. When required to create an alias, just add the AS alias clause after the name of the table, column or expression in the select list or the from list. AS keyword is optional, and the user can specify alias directly after the original name. If the alias or other identifier has the same name as the internal keyword, you need to add the `` symbol to the name. Aliases are sensitive to the lowercase and uppercase letters.

Example:

```
  mysql> select tiny_column as name, int_column as sex from big_table;
mysql> select sum(tiny_column) as total_count from big_table;
mysql> select one.tiny_column, two.int_column from small_table one, big_table two \
    -> where one.tiny_column = two.tiny_column;
```

SQL Statement

DDL Statement

Create Database

This statement is used to create a database.

Syntax:

```
 CREATE DATABASE [IF NOT EXISTS] db_name;
```

Example:

```
 Create a database db_test
CREATE DATABASE db_test;
```

Create Table

This statement is used to create a table.

Syntax:

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [database.]table_name
(column_definition[, column_definition, ...])
[ENGINE = [olap|mysql|broker]]
[key_desc]
[partition_desc]
[distribution_desc]
[PROPERTIES ("key"="value", ...)]
[BROKER PROPERTIES ("key"="value", ...)];
```

Column_definition

Syntax:

```
col_name col_type [agg_type] [NULL | NOT NULL] [DEFAULT "default_value"]
```

- Col_name: Column name
- Col_type: Column type, which can be INT, DOUBLE, DATE, etc. see the Data Type section.
- Agg_type: Means aggregation type. Five aggregation types are supported, including: SUM, MAX, MIN, REPLACE and HLL_UNION. Among them, the HLL_UNION is only used in HLL column, which is the unique aggregation method for HLL. Aggregation type is optional. If it is not specified, it means that the column is a dimension column (key column), otherwise it is a fact column (value column). In tabling statements, all key columns must precede the value column. A table can have no value column, so it is a dimension table; however, it must have a key column. This type is only useful for aggregation models (the type of key_desc is AGGREGATE KEY). Other models do not need to specify this type.
- NULL allowed or not: NULL is allowed by default and is expressed by \N when importing

Note:

During import, Palo automatically merges the value columns corresponding to the same key column according to the specified aggregation method (for aggregation models). For example, a table in Palo contains three columns: k1, k2 and v, of which v is a value column of type int, and if the aggregation method is SUM, k1 and k2 are key columns. If the original data are as follows:

```
 | k1  | k2  | v   |
|-----|-----|-----|
| 1   | 1   | 10  |
| 1   | 2   | 20  |
| 2   | 2   | 30  |
```

The newly imported data are as follows:

```
 | k1  | k2  | v   |
|-----|-----|-----|
| 1   | 1   | 5   |
| 2   | 2   | 10  |
| 3   | 1   | 5   |
```

After being imported, the data in Palo is as follows

```
 | k1 | k2 | v  |
|-----|-----|-----|
| 1  | 1  | 15 |
| 1  | 2  | 20 |
| 2  | 2  | 40 |
| 3  | 1  | 5  |
```

It can be seen that when k1 and k2 are the same, column v is aggregated using SUM aggregation method.

ENGINE type

Note:

ENGINE defaults to olap, which means that Palo provides storage support. mysql and broker can also be selected.

Mysql type is used to store dimension tables, which are maintained by users themselves for easy modification. During querying, Palo can automatically connect mysql tables and olap tables. The following properties information is required for use of mysql type.

```
 PROPERTIES (
"host" = "mysql_server_host",
"port" = "mysql_server_port",
"user" = "your_user_name",
"password" = "your_password",
"database" = "database_name",
"table" = "table_name"
)
```

The "table_name" in the "table" entry is the real table name in mysql. The table_name in the CREATE TABLE statement is the name of the mysql table in Palo, and the two may be different.

The broker type indicates that access to the table is performed via through the specified broker, and the following information needs to be provided in properties

```
 PROPERTIES (
"broker_name" = "broker_name",
"paths" = "file_path1[,file_path2]",
"column_separator" = "value_separator",
"line_delimiter" = "value_delimiter"
)
```

In addition, Property information required by Broker can be provided and passed through BROKER PROPERTIES. For example, HDFS needs to be passed in

```
 BROKER PROPERTIES(
"username" = "name",
"password" = "password"
)
```

This requires different content to be passed in according to different Broker types.

If there are multiple files in "paths", they are separated by commas [,]. If the file name contains commas, it is replaced by %2c. If the file name contains%, it is replaced by %25. Currently, the file content format supports CSV and GZ, BZ2, LZ4, and LZO(LZOP) compression formats.

key_desc

Syntax:

```
key_type(k1[,k2 ...])
```

Notes:

The data are sorted according to the specified key column and have different characteristics according to different key_type.

Key_type supports the following types:

- AGGREGATE KEY: For the same records in the key column, they are aggregated in the value column according to the specified aggregation type, which is suitable for business scenarios such as reports and multidimensional analysis.
- Unique key: For the records in the same key column, they are overwritten in the value column according to the import order, which is suitable for point query services that are added, deleted or modified by the key column.
- Duplicate Key: The same records in the key column exist in Palo at the same time, which is suitable for business scenarios where detailed data or data has no aggregation characteristics.

partition_desc

Syntax:

```
 PARTITION BY RANGE (k1)
(
PARTITION partition_name VALUES LESS THAN MAXVALUE|("value1") [("key"="value")],
PARTITION partition_name VALUES LESS THAN MAXVALUE|("value2") [("key"="value")],
...
)
```

Partition uses the specified key column and the specified data range to partition the data, and each partition physically corresponds to a different data block, which facilitates fast filtering and deletion by partition. Currently, it only supports partitioning by Range. There can only be one partition column, which must be a key column. Note that no comma is inserted after the last PARTITION clause.

Notes:

- Partition name only supports the form in which it starts with a letter and can only consist of letters, numerals and underscore.
- Currently, only the following types of columns are supported as partition column, and only one partition column TINYINT, SAMLLINT, INT, BIGINT, LARGEINT, DATE, DATETIME can be specified.
- Partitions are left closed and right open, and the left boundary of the first partition is taken as the minimum value.
- If a partition is specified, imported data that cannot be determined as a partition range will be filtered out.
- The key-value pair after each partition can set some attribute of the partition. Currently, the following attributes are supported:
  - Storage_medium: It is used to specify the initial storage medium for this partition, i.e., SSD or HDD. The default medium is HDD. Single-node SSD capacity is 50G, and storage medium can be selected according to performance requirements and data volume.
  - Storage_cooldown_time: When the storage medium is set to SSD, specify the storage expiration time of the partition on SSD. The default storage time is 7 days. The format is "yyyy-MM-dd HH:mm:ss”. After expiration, the data will be automatically migrated to HDD.
  - Replication_num: It specifies the number of replicas for the partition. The default is 3

distribution_desc

Distribution is used to specify how to divide barrels. You can choose Random and Hash to divide barrels.

Random bucketing syntax:

```
DISTRIBUTED BY RANDOM [BUCKETS num]
```

Hash-bucketing syntax:

```
DISTRIBUTED BY HASH (k1[,k2 ...]) [BUCKETS num]
```

Notes:

- Random uses all key columns for hash-bucketing, with the default number of partitions being 10. Hash uses the specified key column for bucketing, and the default number of partitions is 10. If the ENGINE type is olap, you must specify a bucketing method; in case of Mysql, it is not required to specify the bucketing method.
- It is recommended not to use Random bucketing method, but to use Hash-bucketing method.

properties

If the ENGINE type is olap, you can specify row storage or column storage in properties

If the ENGINE type is olap and partition information is not specified, you can set storage media, storage expiration time, number of replicas and other attributes in properties. If partition information is specified, attribute values need to be specified separately for each partition. See partition_desc.

```
 PROPERTIES (
"storage_medium" = "[SSD|HDD]",
["storage_cooldown_time" = "yyyy-MM-dd HH:mm:ss"],
["replication_num" = "3"]
)
```

If the ENGINE type is olap and the storage_type is column, you can specify a column to use the bloom filter index. Bloom filter index is only applicable when the query criteria are in and equal, and the more scattered the values in the column, the better the effect. Currently, only columns with the following conditions are supported: key columns other than TINYINT FLOAT DOUBLE type and value columns whose aggregation method is REPLACE

```
 PROPERTIES (
"bloom_filter_columns"="k1,k2,k3"
)
```

Supplementary notes on table building

- Notes on Partition and Distribution:
    - Palo supports composite partitioning, the first level is called partition, which corresponds to the partition_desc clause in the tabling statement. The second level is called Distribution, which corresponds to the distribution_desc clause in the table-building statement. Partition is optional. When Partition is not specified during tabling, the system automatically creates a unique Partition. Distribution must be explicitly specified. It is recommended to create a Partition in the following scenarios:
    - Historical data deletion requirement: If it is required to delete historical data (for example, only the data for the last N days are maintained). Using composite partitions, you can achieve your goal by deleting historical partition.

- To solve the data skew problem: specify the number of buckets in each partition, separately. For example, partitioning by day, i.e., when the data size per day varies significantly, the data in different partitions can be reasonably divided by specifying the number of buckets in the partition.
- If there are business requirements, such as data division, import, query, deletion and historical data backtracking according to time dimension, it is recommended to use composite partitioning function.

- Reasonable table mode:

A prefix index-like structure is used in Palo to improve query performance. Data is sorted by key column within Palo and organized into Data Blocks. The first few columns of the first row of each Data Block are used as indexes for this Data Block and are created when data is imported. This index can help Palo filter some Data Blocks quickly. Considering factors such as index size, Palo uses up to the first 36 bytes of a row as an index, and interrupts when VARCHAR type is encountered, and VARCHAR type only uses up to the first 20 bytes of a string. The following is an example.

Schema of Table 1:

Field	Type	Agg
k1	int	
k2	bigint	
k3	char(24)	
k4	int	
pv	bigint	SUM

The length sum of the first three columns is (4+8+24=)36, which is exactly 36 bytes, so the first three columns are used as prefix indexes.

Schema of Table 2:

Field	Type	Agg
k1	int	
k2	bigint	
k3	varchar(30)	
k4	int	
pv	bigint	SUM

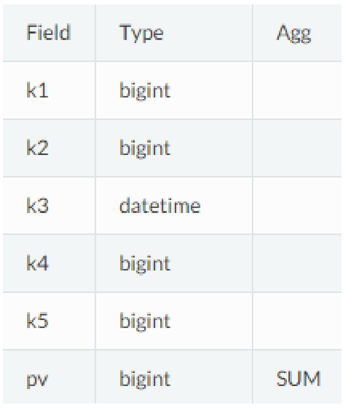
The length of the first two columns is (4+8=)12, which does not reach 36, but the third column is varchar, so the first three columns are used as indexes, with only the first 20 bytes removed in k3.

Schema of Table 3:

Field	Type	Agg
k3	varchar(30)	
k1	int	
k2	bigint	
k4	int	
pv	bigint	SUM

The first column of the table is of varchar type, so only the first 20 bytes of column k3 are used as indexes.

Schema of Table 4:



Field	Type	Agg
k1	bigint	
k2	bigint	
k3	datetime	
k4	bigint	
k5	bigint	
pv	bigint	SUM

The sum of the lengths of the first four columns is (8+8+8+8=)32. If the fifth column (8 bytes) is added, it exceeds 36 bytes. Thus, only the first four columns are used as indexes.

If the same statement is executed for Tables 2 and 3:

```
SELECT * from tbl WHERE k1 = 12345;
```

The performance of Table 2 is superior to Table 3, because the k1 index can be used in Table 2, while Table 3 only has k3 as the index, and the query scans the entire table. Therefore, when creating a table, you should try to put the columns that are frequently used and highly selective in the front part, try not to put varchar type in the first few columns, and use integer type as the index column.

Example:

1. Create an olap table, use Random bucketing method and column storage, and aggregate records with the same key.

   ```
   CREATE TABLE example_db.table_random
   (
   k1 TINYINT,
   k2 DECIMAL(10, 2) DEFAULT "10.5",
   v1 CHAR(10) REPLACE,
   v2 INT SUM
   )
   ENGINE=olap AGGREGATE KEY(k1, k2)
   DISTRIBUTED BY RANDOM BUCKETS 32
   PROPERTIES ("storage_type"="column");
   ```

2. Create an olap table, use Hash-bucketing method and row storage, and overwrite records with the same key. Set the initial storage medium and storage expiration time.

   ```
   CREATE TABLE example_db.table_hash
   (
   k1 BIGINT,
   k2 LARGEINT,
   v1 VARCHAR(2048),
   v2 SMALLINT DEFAULT "10"
   )
   ENGINE=olap UNIQUE KEY(k1, k2)
   DISTRIBUTED BY HASH (k1, k2) BUCKETS 32
   PROPERTIES
   "storage_type"="row" ,
   "storage_medium" = "SSD",
   ```

```
"storage_cooldown_time" = "2015-06-04 00:00:00"
);
```

3. Create an olap table, use Key Range partition and Hash-bucketing. Column storage is used by default. Records with the same key exist simultaneously. Set the initial storage medium and storage expiration time.

```
CREATE TABLE example_db.table_range
(
k1 DATE,
k2 INT,
k3 SMALLINT,
v1 VARCHAR(2048),
v2 DATETIME DEFAULT "2014-02-04 15:36:00"
)
ENGINE=olap DUPLICATE KEY(k1, k2, k3)
PARTITION BY RANGE (k1)
(
PARTITION p1 VALUES LESS THAN ("2014-01-01")
("storage_medium" = "SSD", "storage_cooldown_time" = "2015-06-04 00:00:00"),
PARTITION p2 VALUES LESS THAN ("2014-06-01")
("storage_medium" = "SSD", "storage_cooldown_time" = "2015-06-04 00:00:00"),
PARTITION p3 VALUES LESS THAN ("2014-12-01")
)
DISTRIBUTED BY HASH(k2) BUCKETS 32;
```

Note:

This statement divides the data into the following 3 partitions:

```
 ( { MIN }, {"2014-01-01"} )
[ {"2014-01-01"}, {"2014-06-01"} )
[ {"2014-06-01"}, {"2014-12-01"} )
```

Data outside these partitions are filtered as invalid data.

4. Create a mysql table

```
CREATE TABLE example_db.table_mysql
(
k1 DATE,
k2 INT,
k3 SMALLINT,
k4 VARCHAR(2048),
k5 DATETIME
)
ENGINE=mysql
PROPERTIES
(
"host" = "127.0.0.1",
"port" = "8239",
"user" = "mysql_user",
"password" = "mysql_passwd",
"database" = "mysql_db_test",
"table" = "mysql_table_test"
```

)

5. Use the " Partition , " data to create a broker external table with data files stored on HDFS

   CREATE EXTERNAL TABLE example_db.table_broker (k1 DATE, k2 INT, k3 SMALLINT, k4 VARCHAR(2048), k5 DATETIME) ENGINE=broker PROPERTIES "broker_name" = "hdfs", "path" = "hdfs://hdfs_host:hdfs_port/data1,hdfs://hdfs_host:hdfs_port/data2,hdfs://hdfs_host:hdfs_port/data3%2c4", column_separator: "line_delimiter" = "\n") BROKER PROPERTIES ("username" = "hdfs_user", "password" = "hdfs_password")

6. Create a table containing HLL column

   CREATE TABLE example_db.example_table (k1 TINYINT, k2 DECIMAL(10, 2) DEFAULT "10.5", v1 HLL HLL_UNION, v2 HLL HLL_UNION) ENGINE=olap AGGREGATE KEY(k1, k2) DISTRIBUTED BY RANDOM BUCKETS 32 PROPERTIES ("storage_type"="column");

Drop Database

This statement is used to delete the database

Syntax:

```
DROP DATABASE [IF EXISTS] db_name;
```

Example:

Delete the database db_test

```
DROP DATABASE db_test;
```

Drop Table

This statement is used to delete a table

Syntax:

```
DROP TABLE [IF EXISTS] [db_name.]table_name;
```

Example:

1. Delete a table

   DROP TABLE my_table;

2. If it exists, delete the table of the specified database

   DROP TABLE IF EXISTS example_db.my_table;

Alter Database

This statement is used to set the quota for the specified database. (Administrator Only)

Syntax:

```
ALTER DATABASE db_name SET DATA QUOTA quota;
```

Example:

Set the specified database data quota to 1GB

```
ALTER DATABASE example_db SET DATA QUOTA 1073741824;
```

Alter Table

This statement is used to modify the existing table. The statement is divided into three types of operations: partition, rollup, and scheme change. Partition is the first-level partition in the above-mentioned composite partition; rollup is an operation related to materialized indexes; and schema change is used to modify the table structure. These three operations cannot appear in an ALTER TABLE statement at the same time. Among them, scheme change and rollup are asynchronous operations. Where the task is submitted successfully, it is returned. Afterwards, you can use SHOW ALTER command to view the progress. Partition is a synchronous operation, and a return command is given to indicate that the execution is complete.

Syntax:

```
ALTER TABLE [database.]table alter_clause1[, alter_clause2, ...];
```

Alter_clause is divided into four types: partition, rollup, scheme change and rename.

Partition-supported operations

Add Partitions

Syntax:

```
ADD PARTITION [IF NOT EXISTS] partition_name VALUES LESS THAN [MAXVALUE|("value1")] ["key"="value"]
[DISTRIBUTED BY RANDOM [BUCKETS num] | DISTRIBUTED BY HASH (k1[,k2 ...]) [BUCKETS num]]
```

Note:

- The partition is left closed and right open. The user specifies the right boundary, and the system automatically determines the left boundary
- If no bucketing method is specified, the bucketing method used for creating a table is automatically used.
- If a bucketing method has been specified, only the number of buckets can be modified, but the bucketing method or columns cannot be modified.
- Some attributes of the partition can be set in the ["key"="value"] section. for specific instructions, see CREATE TABLE

Delete Partition

Syntax:

```
DROP PARTITION [IF EXISTS] partition_name
```

Note:

- Tables that are partitioned must maintain at least one partition
- After executing DROP PARTITION for a period, the deleted partition can be recovered by the RECOVER statement. For detailed information, see the RECOVER statement.

Modify partition properties

Syntax:

```
MODIFY PARTITION partition_name SET ("key" = "value", ...)
```

Note:

- Currently, modification of the storage_medium, storage_cooldown_time and replication_num attributes of partitions is supported.
- When no partition is specified when creating the table, partition_name is the same as the table name.

Rollup-supported operations

Rollup index is similar to materialized views. There is no rollup index in this table after the table is created, and only one base index exists, whose name is as same as the table's name. The user can create one or more rollup indexes for a table, and each rollup index contains a subset of key and value columns in base index, for which the Palo generates independent data to improve query performance. If all the columns involved in a query are included in a rollup index, Palo selects to scan the rollup index instead of all the data. The user can select to create Rollup Index according to the characteristics of his/her application, and the operations supported by Rollup:

Create rollup index

Syntax:

```
  ADD ROLLUP rollup_name (column_name1, column_name2, ...) [FROM from_index_name] [PROPERTIES ("key"="value",
...)]
```

Note:

- If from_index_name is not specified, it is created from the base index by default.
- The column in the rollup table must be an existing column in from_index
- In properties, you can specify the storage format. For detailed information, see CREATE TABLE.

Delete rollup index

Syntax:

```
  DROP ROLLUP rollup_name
 [PROPERTIES ("key"="value", ...)]
```

Note:

- Base index cannot be deleted
- The deleted rollup index can be recovered by the RECOVER statement within a period when the DROP ROLLUP is executed. For detailed information, see the RECOVER statement.

schema change

The Schema change is used to modify the table structure, including adding columns, deleting columns, modifying column types, and adjusting column order. You can modify the structure of base index and rollup index.

Operations supported by scheme change:

Adds a column to the specified location of the specified index

Syntax:

```
  ADD COLUMN column_name column_type [KEY | agg_type] [DEFAULT "default_value"]
[AFTER column_name|FIRST]
[TO index_name]
[PROPERTIES ("key"="value", ...)]
```

Note:

- If the value column is added to the aggregation model, it is required to specify the agg_type.
- If you add a key column to a non-aggregate model, it is required to specify the KEY keyword
- Columns that already exist in base index cannot be added to rollup index. If required, you can recreate a rollup index.

Adds multiple columns to the specified index

Syntax:

```
  ADD COLUMN (column_name1 column_type [KEY | agg_type] DEFAULT "default_value", ...)
[TO index_name]
[PROPERTIES ("key"="value", ...)]
```

Note:

- If the value column is added to the aggregation model, it is required to specify the agg_type.
- If you add a key column to a non-aggregate model, it is required to specify the KEY keyword
- You cannot add columns that already exist in base index to rollup index. If required, you can recreate a rollup index.

Deletes a column from the specified index

Syntax:

```
  DROP COLUMN column_name [FROM index_name]
```

Note:

- Partition column cannot be deleted
- If a column is deleted from the base index, and if it is included in the rollup index, it is also deleted.

Modify the column type and column position of the specified index

Syntax:

```
  MODIFY COLUMN column_name column_type [KEY | agg_type] [DEFAULT "default_value"]
[AFTER column_name|FIRST]
[FROM index_name]
[PROPERTIES ("key"="value", ...)]
```

Note:

- If you modify the value column with aggregation type, it is required to specify the agg_type.
- If you modify a key column with non-aggregate type, it is required to specify the KEY keyword.
- Only the type of the column can be modified, and other attributes of the column remain the same, i.e., other attributes should be explicitly written out in the statement according to the original attributes. See example 5 for schema change.
- Partition columns cannot be modified

- The following types of conversions are currently supported. Loss of accuracy is guaranteed by the user. TINYINT, SMALLINT, INT, BIGINT is converted to TINYINT, SMALLINT, INT, BIGINT, DOUBLE. LARGEINT is converted to DOUBLE. VARCHAR supports modification of the maximum length.
- Change from NULL to NOT NULL is not supported.

Resort the columns for the specified index

Syntax:

```
 ORDER BY (column_name1, column_name2, ...)
[FROM index_name]
[PROPERTIES ("key"="value", ...)]
```

Note:

- All columns in index must be written out
- Value column follows key column

rename

The Rename operation is used to modify the table name, rollup index name, and partition name.

Rename-supported operations:

Modify table name

Syntax:

```
RENAME new_table_name
```

Modify rollup index name

Syntax:

```
RENAME ROLLUP old_rollup_name new_rollup_name
```

Modify partition name

Syntax:

```
RENAME PARTITION old_partition_name new_partition_name
```

Example:

1. Add a partition to existing partition [MIN, 2013-01-01), and partition [2013-01-01, 2014-01-01), and use the default bucketing method.

   ALTER TABLE example_db.my_table
   ADD PARTITION p1 VALUES LESS THAN ("2014-01-01");

2. Add a partition and use a new bucketing method.

   ALTER TABLE example_db.my_table
   ADD PARTITION p1 VALUES LESS THAN ("2015-01-01")
   DISTRIBUTED BY RANDOM BUCKETS 20;

3. Delete a partition

   ```
   ALTER TABLE example_db.my_table
   DROP PARTITION p1;
   ```

4. Create an index: For the example_rollup_index, it is based on the base index(k1,k2,k3,v1,v2), and column storage is selected.

   ```
   ALTER TABLE example_db.my_table
   ADD ROLLUP example_rollup_index(k1, k3, v1, v2)
   PROPERTIES("storage_type"="column");
   ```

5. Create an index: For the example_rollup_index2, it is based on example _ rollup _ index (k1, k3, v1, v2).

   ```
   ALTER TABLE example_db.my_table
   ADD ROLLUP example_rollup_index2 (k1, v1)
   FROM example_rollup_index;
   ```

6. Delete an index: example_rollup_index2

   ```
   ALTER TABLE example_db.my_table
   DROP ROLLUP example_rollup_index2;
   ```

7. Add a key column new_col (non-aggregation model) after col1 of the example_rollup_index.

   ```
   ALTER TABLE example_db.my_table
   ADD COLUMN new_col INT KEY DEFAULT "0" AFTER col1
   TO example_rollup_index;
   ```

8. Add a value column new_col (non-aggregation model) after col1 of the example_rollup_index

   ```
   ALTER TABLE example_db.my_table
   ADD COLUMN new_col INT DEFAULT "0" AFTER col1
   TO example_rollup_index;
   ```

9. Add a key column new_col (aggregation model) after col1 of the example_rollup_index

   ```
   ALTER TABLE example_db.my_table
   ADD COLUMN new_col INT DEFAULT "0" AFTER col1
   TO example_rollup_index;
   ```

10. Add a value column new_col SUM aggregation type (aggregation model) after col1 of example_rollup_index

    ```
    ALTER TABLE example_db.my_table
    ADD COLUMN new_col INT SUM DEFAULT "0" AFTER col1
    TO example_rollup_index;
    ```

11. Add multiple columns to the example_rollup_index (aggregation model)

    ```
    ALTER TABLE example_db.my_table
    ADD COLUMN (col1 INT DEFAULT "1", col2 FLOAT SUM DEFAULT "2.3")
    TO example_rollup_index;
    ```

12. Delete a column from the example_rollup_index

    ```
    ALTER TABLE example_db.my_table
    DROP COLUMN col2
    FROM example_rollup_index;
    ```

13. Modify the type of col1 column of the base index to BIGINT and move it to the position after col2 column

    ```
    ALTER TABLE example_db.my_table
    MODIFY COLUMN col1 BIGINT DEFAULT "1" AFTER col2;
    ```

14. Modify the maximum length of val1 column of base index. The original val1 is (val1 VARCHAR (32) REPLACEDDEFAULT "abc")

    ALTER TABLE example_db.my_table
    MODIFY COLUMN val1 VARCHAR(64) REPLACE DEFAULT "abc";

15. Resort the columns in example_rollup_index (set the original column sequence as k1,k2,k3,v1, and v2)

    ALTER TABLE example_db.my_table
    ORDER BY (k3,k1,k2,v2,v1)
    FROM example_rollup_index;

16. Two operations are performed simultaneously

    ALTER TABLE example_db.my_table
    ADD COLUMN v2 INT MAX DEFAULT "0" AFTER k2 TO example_rollup_index,
    ORDER BY (k3,k1,k2,v2,v1) FROM example_rollup_index;

17. Modify the bloom filter column of the table

    ALTER TABLE example_db.my_table PROPERTIES ("bloom_filter_columns"="k1,k2,k3");

18. Modify the table named table1 to table2

    ALTER TABLE table1 RENAME table2;

19. Change the rollup index named rollup1 in table example_table to rollup2.

    ALTER TABLE example_table RENAME ROLLUP rollup1 rollup2;

20. Modify the partition named p1 in table example_table to p2

    ALTER TABLE example_table RENAME PARTITION p1 p2;

Cancel Alter

This statement is used to undo an Alter operation

Revoke Alter table column(schema change) syntax:

```
CANCEL ALTER TABLE COLUMN FROM db_name.table_name
```

Undo Alter table rollup operation

```
CANCEL ALTER TABLE ROLLUP FROM db_name.table_name
```

Example:

1. Undo ALTER COLUMN operation for my_table.

   CANCEL ALTER TABLE COLUMN
   FROM example_db.my_table;

2. Undo the ADD ROLLUP operation under my_table.

   CANCEL ALTER TABLE ROLLUP
   FROM example_db.my_table;

DML Statement

Load

This statement is used to import data into the specified table. This operation updates the data of both base index and rollup index related to this table. This is an asynchronous operation and will be returned if the task is submitted successfully. After execution, you can use the SHOW LOAD command to view the progress.

NULL import is represented by \N. To convert other strings to NULL, you can use replace_value to convert.

Syntax:

```
  LOAD LABEL load_label
(
data_desc1[, data_desc2, ...]
)
broker
opt_properties
```

load_label

Load_label is the label of the current import batch, which is specified by the user. It is required to guarante its uniqueness in a database. In other words, the label that was successfully imported in a certain database cannot be reused in this database. This label is used to uniquely determine an import in the database to facilitate management and query.

Syntax:

```
  [database_name.]your_label
```

data_desc

It is used to specifically describe a batch of imported data.

Syntax:

```
  DATA INFILE
(
"file_path1 [, file_path2, ...]
)
[NEGATIVE]
INTO TABLE table_name
[PARTITION (p1, p2)]
[COLUMNS TERMINATED BY "column_separator"]
[(column_list)]
[SET (k1 = func(k2))]
```

Note:

- File_path, the path of a file in broker, can be assigned to a file or all files in a directory with the /* wildcard character.
- NEGATIVE: If this parameter is specified, it is equivalent to importing a batch of "negative" data. It is used to offset the same batch of data that are previously imported. This parameter is only applicable when the value column exists and the aggregation type of the value column is SUM. Broker import is not supported
- PARTITION: If this parameter is specified, only the specified partition is imported, and data other than the imported partition is filtered out. If not specified, all partitions of table are imported by default.
- Column_separator: It is used to specify the column separator in the import file. The default is \t. If it is an invisible character, you need to prefix it with \\x and use hexadecimal system to represent the separator. For example, the separator \x01 of the hive file is assigned as "\\x01”
- Column_list: It is used to specify the correspondence between columns in the imported file and columns in table. When it is necessary to skip a column in the imported file, specify the column as a column name that does not exist in table. Its syntax is as follows:

(col_name1, col_name2, ...)

- SET: If you specify this parameter, you can convert a column of the source file according to the function, and then import the converted result into the table. The currently supported functions are:
    - Strftime(fmt, column) date conversion function
        - fmt: Date format, like %Y%m%d%H%M%S (month, year, day, hour, minute and second)
        - column: The column in column_list, i.e., the column in the input file. The storage content should be a digital timestamp. If there is no column_list, the columns of the input file are defaulted according to the column order of the palo table.
    - Time _ format (output _ fmt, input _ fmt, column) date format conversion
        - output_fmt: The converted date format is %Y%m%d%H%M%S (month, year, day, hour, minute and second)
        - input_fmt: The date format of the column before conversion is %Y%m%d%H%M%S (month, year, day, hour, minute and second)
        - column: The column in column_list, i.e., the column in the input file. The storage content should be a date string in input_fmt format. If there is no column_list, the columns of the input file are defaulted according to the column order of the palo table.
    - Alignment _ timestamp (precision, column) aligns timestamps to the specified precision
        - precision: yearmonth
        - column: The column in column_list, i.e., the column in the input file. The storage content should be a digital timestamp. If there is no column_list, the columns of the input file are defaulted according to the column order of the palo table.
        - Note: When the alignment precision is year and month, only timestamps wihin the range of 20050101 ~ 20191231 are supported.
    - Default_value(value) sets the default value of a column imported; if not, the default value of the column is used when the table is created.
    - Md5sum(column1, column2, ...) calculates md5sum from the value of the specified import column and returns a 32-bit hexadecimal string
    - Replace _ value (old_value [,new_value]) replaces the old _ value specified in the import file with a new _ value. If new_value is not specified, the default value of the column is used when the table is created.
    - Hll_hash(column) is used to convert a column in a table or data into a data structure of HLL columns

broker

It is used to specify the Broker used for import.

Syntax:

```
WITH BROKER broker_name ("key"="value"[,...])
```

You need to specify the specific Broker name and the required Broker attributes here.

opt_properties

It is used to specify some special parameters.

Syntax:

```
[PROPERTIES ("key"="value", ...)]
```

You can specify the following parameters:

- Timeout: It specifies the timeout period for the import operation. By default, it is not allowed to exceed the timeout period. Unit: second
- Max_filter_ratio: Maximum tolerable percentage of filterable data (due to non-standard data, etc.). By default, zero tolerance is specified.
- Load_delete_flag: It specifies whether the import deletes data by importing the key column, which is only applicable for UNIQUE KEY. During import, the value column may not be specified. By default, it is false (it is not allowed to support importing by broker)
- Exe_mem_limit: It takes effect in Broker load mode, and specifies the maximum memory available to the backend when the import is executed.

Example:

1. Import a batch of data and specify individual parameters

```
  LOAD LABEL example_db.label1
(
DATA INFILE("hdfs://hdfs_host:hdfs_port/user/palo/data/input/file")
INTO TABLE my_table
)
WITH BROKER hdfs ("username"="hdfs_user", "password"="hdfs_password")
PROPERTIES
(
"timeout"="3600",
"max_filter_ratio"="0.1",
);
```

2. Import a batch of data, including multiple files. Import a different table, specify separator and column correspondence.

```
 LOAD LABEL example_db.label2
(
DATA INFILE("hdfs://hdfs_host:hdfs_port/user/palo/data/input/file1")
INTO TABLE my_table_1
COLUMNS TERMINATED BY ","
(k1, k3, k2, v1, v2),
DATA INFILE("hdfs://hdfs_host:hdfs_port/user/palo/data/input/file2")
INTO TABLE my_table_2
COLUMNS TERMINATED BY "\t"
(k1, k2, k3, v2, v1)
)
WITH BROKER hdfs ("username"="hdfs_user", "password"="hdfs_password");
```

3. Import a batch of data, specify the default separator \\x01 for hive, and use wildcard * to specify all files in the directory

```
LOAD LABEL example_db.label3
(
DATA INFILE("hdfs://hdfs_host:hdfs_port/user/palo/data/input/*")
NEGATIVE
INTO TABLE my_table
COLUMNS TERMINATED BY "\\x01"
)
WITH BROKER hdfs ("username"="hdfs_user", "password"="hdfs_password");
```

4. Import a batch of “negative” data

```
LOAD LABEL example_db.label4
(
DATA INFILE("hdfs://hdfs_host:hdfs_port/user/palo/data/input/old_file")
NEGATIVE
INTO TABLE my_table
COLUMNS TERMINATED BY "\t"
)
WITH BROKER hdfs ("username"="hdfs_user", "password"="hdfs_password");
```

5. Import a batch of data and specify partitions

```
LOAD LABEL example_db.label5
(
DATA INFILE("hdfs://hdfs_host:hdfs_port/user/palo/data/input/file")
INTO TABLE my_table
PARTITION (p1, p2)
COLUMNS TERMINATED BY ","
(k1, k3, k2, v1, v2)
)
WITH BROKER hdfs ("username"="hdfs_user", "password"="hdfs_password");
```

6. Import a batch of data, specify partitions, and make some transformations to the columns of the imported file, as follows:

- K1 converts the tmp_k1 timestamp column into datetime data
- K2 converts tmp_k2 date type data into datetime data
- K3 converts tmp_k3 timestamp column into day-level timestamp
- K4 specifies that the default value is 1 during import
- K5 calculates md5 string from tmp_k1, tmp_k2 and tmp_k3 columns

The import statement is:

```
  LOAD LABEL example_db.label6
(
DATA INFILE("hdfs://hdfs_host:hdfs_port/user/palo/data/input/file")
INTO TABLE my_table
PARTITION (p1, p2)
COLUMNS TERMINATED BY ","
(tmp_k1, tmp_k2, tmp_k3, v1, v2)
SET (
  k1 = strftime("%Y-%m-%d %H:%M:%S", tmp_k1)),
  k2 = time_format("%Y-%m-%d %H:%M:%S", "%Y-%m-%d", tmp_k2)),
  k3 = alignment_timestamp("day", tmp_k3),
  k4 = default_value("1"),
  k5 = md5sum(tmp_k1, tmp_k2, tmp_k3)
)
)
WITH BROKER hdfs ("username"="hdfs_user", "password"="hdfs_password");
```

7. Import data into a table containing an HLL column, which may be a column in the table or data

   LOAD LABEL example_db.label7 (DATA INFILE("hdfs://hdfs_host:hdfs_port/user/palo/data/input/file") INTO TABLE my_table PARTITION (p1, p2) COLUMNS TERMINATED BY "," SET (v1 = hll_hash(k1), v2 = hll_hash(k2))) WITH BROKER hdfs ("username"="hdfs_user", "password"="hdfs_password");

   LOAD LABEL example_db.label8 (DATA INFILE("hdfs://hdfs_host:hdfs_port/user/palo/data/input/file") INTO TABLE `my_table` PARTITION (p1, p2) COLUMNS TERMINATED BY "," (k1, k2, tmp_k3, tmp_k4, v1, v2) SET (v1 = hll_hash(tmp_k3), v2 = hll_hash(tmp_k4))) WITH BROKER hdfs ("username"="hdfs_user", "password"="hdfs_password");

Small Batch Import

Small-batch import is a new importing method provided by Palo, which enables the user to complete the import without relying on Hadoop. This method submits the task not through MySQL client, but through http protocol. The user sends the import description together with data to Palo through http protocol. After receiving the task successfully, Palo immediately returns the success information to the user, but at this moment, the data is not really imported. The user needs to view the specific import results through the 'SHOW LOAD' command.

Syntax:

```
  curl --location-trusted -u user:passwd -T data.file http://fe.host:port/api/{db}/{table}/_load?label=xxx
```

Parameter description:

- User: If the user is contained in default_cluster, it is the user_name. Otherwise, it is user_name@cluster_name.
- Label: It is used to specify the label imported in this batch and to query the job status later. This parameter must be passed in.
- columns: It is used to describe the corresponding column name in the import file. If it is not passed in, then the column sequence in the file is considered to be the same as the sequence in which the table is created, and the specified method is comma separated, for example: columns=k1,k2,k3,k4
- column_separator: It is used to specify the separator between columns, and the default separator is' \t'. It should be noted that url encoding should be used here, for example, it is required to specify ' \t' as separator, then 'column _ separator =% 09' should be input. It is required to specify '\x01' as separator, then ‘column_separator=%01' should be input.
- max_filter_ratio: It is used to specify the maximum percentage of non-standard data allowed to be filtered. The default value is 0. Filtering is not allowed. The custom specification should be as follows: 'max_filter_ratio=0.2', which means that an error rate of 20% is allowed.

- Hll: It is used to specify the correspondence between hll columns in the data and in the table. Columns in the table and columns specified in the data (if columns are not specified, columns in the data columns may be also other non-HLL columns in the table) are separated by “,”. When specifying multiple HLL columns, they are separated by ":”, for example: 'hll1,cuid:hll2,device’

Example:

1. Import the data from the local file 'testData' into the table of 'testTbl' in the database 'testDb' (the user is contained in defalut_cluster)

   curl --location-trusted -u root:root -T testData http://fe.host:port/api/testDb/testTbl/_load?label=123

2. Import the data from the local file 'testData' into the table of' testTbl' in the database' testDb' (user is from test_cluster)

   curl --location-trusted -u root@test_cluster:root -T testData http://fe.host:port/api/testDb/testTbl/_load?label=123

3. Import the data from the local file 'testData' into the table of 'testTbl' in the database 'testDb', allowing an error rate of 20% (the user is in defalut_cluster)

   curl --location-trusted -u root -T testData http://fe.host:port/api/testDb/testTbl/_load?label=123\$amp;max_filter_ratio=0.2

4. Import the data from the local file 'testData' into the table of 'testTbl' in the database 'testDb', allow an error rate of 20%, and specify the column name of the file (the user is contained in the defalut_cluster)

   curl --location-trusted -u root -T testData http://fe.host:port/api/testDb/testTbl/_load?label=123\$amp;max_filter_ratio=0.2\$amp;columns=k1,k2,k3

5. It is imported by means of streaming (the user is contained in the defalut_cluster)

   seq 1 10 | awk '{OFS="\t"}{print $1, $1 * 10}' | curl --location-trusted -u root -T testData http://fe.host:port/api/testDb/testTbl/_load?label=123

6. Import a table containing a HLL column, which can be column in the table or data and used to generate a HLL column (the user is contained in the defalut_cluster)

   curl --location-trusted -u root -T testData http://host:port/api/testDb/testTbl/_load?label=123\&max_filter_ratio=0.2\&hll=hll_column1,k1:hll_column2,k2

   curl --location-trusted -u root -T testData http://host:port/api/testDb/testTbl/_load?label=123\&max_filter_ratio=0.2\&hll=hll_column1,tmp_k4:hll_column2,tmp_k5\&columns=k1,k2,k3,tmp_k4,tmp_k5

Cancel Load

Cancel load is used to revoke the import job that specifies the load label. This is an asynchronous operation and will be returned upon successful task submission. After submission, you can use the show load command to view the progress.

Syntax:

```
CANCEL LOAD [FROM db_name] WHERE LABEL = "load_label";
```

Example:

Revoke import job of example_db _ test _ load _ label with label on database example _ db

```
CANCEL LOAD FROM example_db WHERE LABEL = "example_db_test_load_label";
```

Export

This statement is used to export the data of the specified table to the specified location. This is an asynchronous operation and will be returned if the task is submitted successfully. After execution, you can use the SHOW EXPORT command to view the progress.

Syntax:

```
  EXPORT TABLE table_name
[PARTITION (p1[,p2])]
TO export_path
[opt_properties]
broker;
```

table_name

Currently, the name of the table to be exported currently supports the export of tables whose engine is olap and mysql.

partition

You can export only some specified partitions of a specified table

export_path

The exported path needs to be a directory. At present, it cannot be exported locally, and needs to be exported to broker.

opt_properties

It is used to specify some special parameters.

Syntax:

```
[PROPERTIES ("key"="value", ...)]
```

You can specify the following parameters:

- Column_separator: It specifies the exported column separator, which is default to \t.
- Line_separator: It specifies the exported row separator, which is default to \n.

broker

It is used to specify the broker used for export.

Syntax:

```
WITH BROKER broker_name ("key"="value"[,...])
```

You need to specify the specific broker name and the broker attributes required.

Example:

1. Export all data in testTbl to hdfs

   EXPORT TABLE testTbl TO "hdfs://hdfs_host:port/a/b/c" WITH BROKER "broker_name" ("username"="xxx", "password"="yyy");

2. Export partitions P1 and P2 in testTbl to hdfs

   EXPORT TABLE testTbl PARTITION (p1,p2) TO "hdfs://hdfs_host:port/a/b/c" WITH BROKER "broker_name" ("username"="xxx", "password"="yyy");

3. Export all data in testTbl to hdfs, with ",” being as column separator

   EXPORT TABLE testTbl TO "hdfs://hdfs_host:port/a/b/c" PROPERTIES ("column_separator"=",") WITH BROKER "broker_name" ("username"="xxx", "password"="yyy");

Delete

This statement is used to delete data in the specified table(base index) partition according to conditions. This operation delete the relevant rollup index data at the same time.

Syntax:

```
 DELETE FROM table_name PARTITION partition_name WHERE
column_name1 op value[ AND column_name2 op value ...];
```

Note:

- Optional types of op include: =, <, >, <=, >=, and !=
- Only conditions on the key column can be specified.
- Conditions can only be in "and" relation. If you want to reach an OR relationship, you need to write the condition in two DELETE statements.
- If no partition is created, partition_name is identical to the table_name.

Note:

- This statement may reduce the query efficiency within a period of time after execution. The degree of impact depends on the number of deletion conditions specified in the statement. The more conditions specified, the greater its impact.

Example:

1. Delete the data row with k1 column value as 3 in my_table partition p1

   DELETE FROM my_table PARTITION p1 WHERE k1 = 3;

2. Delete the data row with k1 column value greater than or equal to 3 and k2 column value "abc" in my_table partition p1

   DELETE FROM my_table PARTITION p1 WHERE k1 >= 3 AND k2 = "abc";

SELECT Statement

The Select statement consists of select, from, where, group by, having, order by, union and other parts. Palo's query statement basically meets SQL92 standard. The following describes the supported Select usage in detail.

Join

Join operation is to merge the data of 2 or more tables and then return the result set of some columns in some of the tables. Currently, Palo supports inner join, outer join, semi join, anti join, and cross join. In the Inner join condition, not only the equivalent join is supported, but also the non-equivalent join. For performance consideration, the equivalent join is recommended. Other joins only support equivalent joins.

Syntax:

```
 SELECT select_list FROM
    table_or_subquery1 [INNER] JOIN table_or_subquery2 |
    table_or_subquery1 {LEFT [OUTER] | RIGHT [OUTER] | FULL [OUTER]} JOIN table_or_subquery2 |
    table_or_subquery1 {LEFT | RIGHT} SEMI JOIN table_or_subquery2 |
    table_or_subquery1 {LEFT | RIGHT} ANTI JOIN table_or_subquery2 |
      [ ON col1 = col2 [AND col3 = col4 ...] |
        USING (col1 [, col2 ...]) ]
    [other_join_clause ...]
    [ WHERE where_clauses ]

SELECT select_list FROM
    table_or_subquery1, table_or_subquery2 [, table_or_subquery3 ...]
    [other_join_clause ...]
WHERE
    col1 = col2 [AND col3 = col4 ...]

SELECT select_list FROM
    table_or_subquery1 CROSS JOIN table_or_subquery2
    [other_join_clause ...]
[ WHERE where_clauses ]
```

Self-Join

Palo supports self-joins, i.e., different parts in one table is joined with each other. For example, different columns of the same table is joined. In fact, there is no special syntax identifier Self-join. The conditions on both sides of the Join in the Self-join come from the same table, and it is required to assign them with different aliases.

Example:

```
 SELECT lhs.id, rhs.parent, lhs.c1, rhs.c2 FROM tree_data lhs, tree_data rhs WHERE lhs.id = rhs.parent;
```

Cartesian product(Cross Join)

Cross join can produce a large number of results. It is prudent to use the Cross join. Even if it is required to use the Cross join, filtering conditions should be used and results should be returned as fewer as possible.

Example:

```
 SELECT * FROM t1, t2;
SELECT * FROM t1 CROSS JOIN t2;
```

Inner join

Inner join is the most common join. The returned result comes from the columns requested by two similar tables. The condition of join is that there are same values contained in the columns of the two tables. If the columns of the two tables have same name, we need to use the full name (table_name.column_name) or give an alias to the column.

Example:

```
 -- The following 3 forms are all equivalent.
SELECT t1.id, c1, c2 FROM t1, t2 WHERE t1.id = t2.id;
SELECT t1.id, c1, c2 FROM t1 JOIN t2 ON t1.id = t2.id;
SELECT t1.id, c1, c2 FROM t1 INNER JOIN t2 ON t1.id = t2.id;
```

Outer join

Outer join returns the rows of the left table or right table or both tables. If there is no matching data in another table, set it to

NULL.

Example:

```
 SELECT * FROM t1 LEFT OUTER JOIN t2 ON t1.id = t2.id;
SELECT * FROM t1 RIGHT OUTER JOIN t2 ON t1.id = t2.id;
SELECT * FROM t1 FULL OUTER JOIN t2 ON t1.id = t2.id;
```

Equal and unequal join

In general, the user uses the equivalent join, which requires the operator of the Join condition to be an equal sign. !, , <, > and other symbols can be used by unequal Join under Join condition. Unequal join can produce a large number of results, which may exceed the memory limit in the calculation process. So, it needs to be used carefully. Unequal join only supports inner join.

Example:

```
 SELECT t1.id, c1, c2 FROM t1 INNER JOIN t2 ON t1.id = t2.id;
SELECT t1.id, c1, c2 FROM t1 INNER JOIN t2 ON t1.id > t2.id;
```

Semi join

Left semi join returns only the rows in the left table that can match the data in the right table. No matter how many rows of data in the right table can be matched, the row in the left table is only returned at most once. The principle of the Right semi join is similar, but the returned data is from the right table.

Example:

```
 SELECT t1.c1, t1.c2, t1.c2 FROM t1 LEFT SEMI JOIN t2 ON t1.id = t2.id;
```

Anti join

Left anti join returns only rows in the left table that cannot match the right table. Right anti join reverses this comparison and returns only the rows in the right table that cannot match the left table.

Example:

```
 SELECT t1.c1, t1.c2, t1.c2 FROM t1 LEFT ANTI JOIN t2 ON t1.id = t2.id;
```

Order by

Order by sorts the result sets by comparing the sizes of one or more columns. Order by is a time- and resource-consuming operation because all data should be sent to a node before sorting, and sorting operation requires more memory than non-sorting operation. If you need to return the first N sorting results, you need to use the LIMIT clause. To limit memory usage, the first 65535 sorting results are returned by default if the user does not specify a LIMIT clause.

Syntax:

```
ORDER BY col [ASC | DESC]
```

By default, the sorting is ASC

Example:

```
mysql> select * from big_table order by tiny_column, short_column desc;
```

Group by

The Group by clause is usually used with aggregate functions, such as COUNT(), SUM(), AVG(), MIN (), and MAX (). Columns specified by Group by do not take part in aggregation operation. A Having clause can be added to the Group by clause to filter the results produced by the aggregate function.

Example:

```
 mysql> select tiny_column, sum(short_column) from small_table group by tiny_column;
+-------------+---------------------+
| tiny_column | sum(`short_column`) |
+-------------+---------------------+
|           1 |                   2 |
|           2 |                   1 |
+-------------+---------------------+
2 rows in set (0.07 sec)
```

Having

The Having clause does not filter the row data in the table, but the results produced by the aggregate function. In general, Having is used with aggregate functions (such as COUNT(), SUM(), AVG(), MIN(), MAX ()) and Group by clauses.

Example:

```
 mysql> select tiny_column, sum(short_column) from small_table group by tiny_column having sum(short_column) = 1;
+-------------+---------------------+
| tiny_column | sum(`short_column`) |
+-------------+---------------------+
|           2 |                   1 |
+-------------+---------------------+
1 row in set (0.07 sec)

mysql> select tiny_column, sum(short_column) from small_table group by tiny_column having tiny_column > 1;
+-------------+---------------------+
| tiny_column | sum(`short_column`) |
+-------------+---------------------+
|           2 |                   1 |
+-------------+---------------------+
1 row in set (0.07 sec)
```

Limit

The Limit clause is used to limit the maximum number of rows of return results. By setting the maximum number of rows of the return results, help Palo optimize memory usage. This clause is mainly used in the following scenarios:

- Returns the top-N query result.
- Want to briefly look at the contents contained in the table.
- It can be used when the data in the table is big enough or the Where clause does not filter too much data.

Instructions for use:

The value of the Limit clause must be a numeric literal constant.

Example:

```
 mysql> select tiny_column from small_table limit 1;
+-------------+
| tiny_column |
+-------------+
|           1 |
+-------------+
1 row in set (0.02 sec)

mysql> select tiny_column from small_table limit 10000;
+-------------+
| tiny_column |
+-------------+
|           1 |
|           2 |
+-------------+
2 rows in set (0.01 sec)
```

Offset

The Offset clause causes the result set to skip the results of the first few rows and return the subsequent results directly. The default starting row of the result set is row 0, so the Offset 0 and no Offset return the same result. In general, the Offset clause takes effect when it is used together with the Order by clause and Limit clause.

Example:

```
 mysql> select varchar_column from big_table order by varchar_column limit 3;
+----------------+
| varchar_column |
+----------------+
| beijing        |
| chongqing      |
| tianjin        |
+----------------+
3 rows in set (0.02 sec)

mysql> select varchar_column from big_table order by varchar_column limit 1 offset 0;
+----------------+
| varchar_column |
+----------------+
| beijing        |
+----------------+
1 row in set (0.01 sec)

mysql> select varchar_column from big_table order by varchar_column limit 1 offset 1;
+----------------+
| varchar_column |
+----------------+
| chongqing      |
+----------------+
1 row in set (0.01 sec)

mysql> select varchar_column from big_table order by varchar_column limit 1 offset 2;
+----------------+
| varchar_column |
+----------------+
| tianjin        |
+----------------+
1 row in set (0.02 sec)
```

Note:

It is allowed to use the Offset syntax without Order by, but the Offset is meaningless at this moment. In such case, only the Limit value is taken and the offset value is ignored. Therefore, where no Order by exists, there is still a result when the Offset exceeds the maximum number of rows in the result set. It is recommended that the user must use the Offset together with the Order by.

Union

The Union clause is used to merge the result set of multiple queries.

Syntax:

```
query_1 UNION [DISTINCT | ALL] query_2
```

Instructions for use:

The effect of using only the Union keyword is the same as the Union disitnct. Because deduplication consumes more memory, query by use of the Union all is faster and consumes less memory. If the user wants to perform the Order by and Limit operations on the returned result set, it is required to put the Union operation in the subquery, and then select it from subquery, as well as finally put the Subgquery and Order by outside the subquery.

Example:

```
 mysql> (select tiny_column from small_table) union all (select tiny_column from small_table);
+-------------+
| tiny_column |
+-------------+
|           1 |
|           2 |
|           1 |
|           2 |
+-------------+
4 rows in set (0.10 sec)

mysql> (select tiny_column from small_table) union (select tiny_column from small_table);
+-------------+
| tiny_column |
+-------------+
|           2 |
|           1 |
+-------------+
2 rows in set (0.11 sec)

mysql> select * from (select tiny_column from small_table union all\
    -> select tiny_column from small_table) as t1 \
    -> order by tiny_column limit 4;
+-------------+
| tiny_column |
+-------------+
|           1 |
|           1 |
|           2 |
|           2 |
+-------------+
4 rows in set (0.11 sec)
```

Distinct

The Distinct operator deduplicates the result set.

Example:

```
mysql> -- Returns the unique values from one column.
mysql> select distinct tiny_column from big_table limit 2;
mysql> -- Returns the unique combinations of values from multiple columns.
mysql> select distinct tiny_column, int_column from big_table limit 2;
```

Distinct can be used together with aggregate functions (usually the count function), and count(disitnct) is used to calculate how many different combinations are contained in single column or multiple columns.

```
mysql> -- Counts the unique values from one column.
mysql> select count(distinct tiny_column) from small_table;
+-------------------------------+
| count(DISTINCT `tiny_column`) |
+-------------------------------+
|                             2 |
+-------------------------------+
1 row in set (0.06 sec)
mysql> -- Counts the unique combinations of values from multiple columns.
mysql> select count(distinct tiny_column, int_column) from big_table limit 2;
```

Palo supports multiple aggregate functions using distinct simultaneously.

```
mysql> -- Count the unique value from multiple aggregation function separately.
mysql> select count(distinct tiny_column, int_column), count(distinct varchar_column) from big_table;
```

Subquery

Subqueries are divided into uncorrelated subqueries and correlated subqueries according to their correlation.

Uncorrelated subquery

Uncorrelated subqueries support [NOT] IN and EXISTS.

Example:

```
SELECT x FROM t1 WHERE x [NOT] IN (SELECT y FROM t2);
SELECT x FROM t1 WHERE EXISTS (SELECT y FROM t2 WHERE y = 1);
```

Correlated subqueries

Correlated subqueries support [NOT] IN and [NOT] EXISTS.

Example:

```
SELECT * FROM t1 WHERE x [NOT] IN (SELECT a FROM t2 WHERE t1.y = t2.b);
SELECT * FROM t1 WHERE [NOT] EXISTS (SELECT a FROM t2 WHERE t1.y = t2.b);
```

Subqueries also support scalar queries. It is divided into uncorrelated scalar query, correlated scalar query and scalar query as parameters of common functions.

Example:

```
1. Irrelevant scalar quantum query, the predicate is = sign. For example: output information forf the person with the highest salary
SELECT name FROM table WHERE salary = (SELECT MAX(salary) FROM table);
2. Irrelevant scalar quantum queries, the predicates are>, <, etc. For example: output information about people who have a higher average salary
SELECT name FROM table WHERE salary > (SELECT AVG(salary) FROM table);
3. Related Scalar Subquery. For example: Output the highest salary information of each department
SELECT name FROM table a WHERE salary = （SELECT MAX(salary) FROM table b WHERE b.department= a.department);
4. The scalar subquery is as parameter of ordinary function.
SELECT name FROM table WHERE salary = abs((SELECT MAX(salary) FROM table));
```

With clause

It is a clause that can be added before a SELECT statement, which is used to define aliases for complex expressions that are referenced multiple times in a SELECT. Similar to the CREATE VIEW, except that the table and column names defined in the clause do not persist after the query ends, and do not conflict with the names in the actual table or VIEW.

The advantages of using the WITH clause include:

1. It is convenient and easy to maintain, reducing the internal duplication of queries.
2. It is easier to read and understand the SQL code by abstracting the most complex part of the query into separate blocks.

Example:

```
-- Define one subquery at the outer level, and another at the inner level as part of the
-- initial stage of the UNION ALL query.
with t1 as (select 1) (with t2 as (select 2) select * from t2) union all select * from t1;
```

SHOW statement

Show alter

This statement is used to show the implementation of various modification tasks that are in progress currently.

Syntax:

```
SHOW ALTER TABLE [COLUMN | ROLLUP] [FROM db_name];
```

Note:

- TABLE COLUMN: It shows ALTER task to modify column
- TABLE ROLLUP: It shows the task of creating or deleting ROLLUP index.
- If the db_name is not specified, the current default db is used.

Example:

1. It shows the task execution of all modified columns for the default db

   SHOW ALTER TABLE COLUMN;

2. It shows the execution of the task of creating or deleting ROLLUP index for the specified db.

   SHOW ALTER TABLE ROLLUP FROM example_db;

Show data

This statement is used to show the data size.

Syntax:

```
SHOW DATA [FROM db_name[.table_name]];
```

Note:

If the FROM clause is not specified, it is used to show the size of data subdivided into table under the current db. If the FROM clause is specified, the size of data subdivided into indexes under table is shown.

Example:

1. Show the data size and summary data size of each table for the default db

   SHOW DATA;

2. Show the subdivision data size of the specified table below the specified db

   SHOW DATA FROM example_db.table_name;

Show databases

This statement is used to show the currently visible database

Syntax:

```
SHOW DATABASES;
```

Show load

This statement is used to show the execution of the specified import task.

Syntax:

```
SHOW LOAD
[FROM db_name]
[
WHERE
[LABEL [ = "your_label" | LIKE "label_matcher"]]
[STATUS = ["PENDING"|"ETL"|"LOADING"|"FINISHED"|"CANCELLED"|]]
]
[ORDER BY ...]
[LIMIT limit];
```

Note:

- If db_name is not specified, the current default db is used.
- If the LABE LIKE is used, the label matching the import task contains the import task of label_matcher
- If LABEL = is used, the specified label is exactly matched.
- If the STATUS is specified, the LOAD status is matched.
- You can use ORDER BY to sort any combination of columns.
- If LIMIT is specified, Limit matching records are shown. Otherwise, all records are shown.

Example:

1. Show all import tasks of default db

   SHOW LOAD;

2. Show the import task of the specified db. The label contains the string "2014_01_02" and shows the oldest 10 strings.

   SHOW LOAD FROM example_db WHERE LABEL LIKE "2014_01_02" LIMIT 10;

3. Show the import task of the specified db, specify the label as "load_example_db_20140102" and sort it in descending order of LoadStartTime.

   SHOW LOAD FROM example_db WHERE LABEL = "load_example_db_20140102" ORDER BY LoadStartTime DESC;

4. Show the import task of the specified db, specify label as "load_example_db_20140102", state as "loading", and sort it in descending order by LoadStartTime

   SHOW LOAD FROM example_db WHERE LABEL = "load_example_db_20140102" AND STATE = "loading" ORDER BY LoadStartTime DESC;

Show export

This statement is used to show the execution of the specified export task

Syntax:

```
 SHOW EXPORT
[FROM db_name]
[
WHERE
[EXPORT_JOB_ID = your_job_id]
[STATE = ["PENDING"|"EXPORTING"|"FINISHED"|"CANCELLED"]]
]
[ORDER BY ...]
[LIMIT limit];
```

Note:

- If db_name is not specified, the current default db is used.
- If STATE is specified, the EXPORT state is matched.
- ORDER BY can be used to sort any combination of columns.
- If LIMIT is specified, Limit matching records are shown. Otherwise, all records are shown.

Example:

1. Show all export tasks of default db

   SHOW EXPORT;

2. Show the export tasks of the specified db, and sort them in descending order by StartTime

   SHOW EXPORT FROM example_db ORDER BY StartTime DESC;

3. Show the export tasks of the specified db in "exporting” state, and sort them in descending order by StartTime

   SHOW EXPORT FROM example_db WHERE STATE = "exporting" ORDER BY StartTime DESC;

4. Show the export task that specifies db and job_id

   SHOW EXPORT FROM example_db WHERE EXPORT_JOB_ID = job_id;

Show partitions

This statement is used to show partition information

Syntax:

```
SHOW PARTITIONS FROM [db_name.]table_name [PARTITION partition_name];
```

Example:

1. Show partition information of the specified table under the specified db

   SHOW PARTITIONS FROM example_db.table_name;

2. Show the information of the specified partition of the specified table under the specified db

   SHOW PARTITIONS FROM example_db.table_name PARTITION p1;

Show Quota

This statement is used to display the resource allocation of different groups of a user.

Syntax:

```
SHOW QUOTA FOR [user]
```

Example:

Displays the allocation of system users' resources in each group.

```
SHOW QUOTA FOR system;
```

Show Resource

This statement is used to display a user's weight on different resources

Syntax:

```
SHOW RESOURCE [LIKE user_name]
```

Example:

It displays the weight of system users on different resources

```
SHOW RESOURCE LIKE "system";
```

Show tables

This statement is used to show all table under the current db.

Syntax:

```
SHOW TABLES;
```

Show tablet

This statement is used to display information related to tablet (for administrators only)

Syntax:

```
SHOW TABLET [FROM [db_name.]table_name | tablet_id]
```

Example:

1. Show all TABLE information of the specified table below the specified db

   SHOW TABLET FROM example_db.table_name;

2. Display the parent level id information of the tablet with the specified tablet id of 10000.

   SHOW TABLET 10000;

Account Management

Create user

This statement is used to create a user and requires administrator privileges. In the event of creating the user under non-default_cluster, the user name is user_name@cluster_name when logging in to connect palo, mini load, etc. In the event of creating the user under default_cluster, the user does not need to add @cluster_name to the user name when logging in to connect palo and mini load, i.e., just use user_name directly.

Syntax:

```
 CREATE USER user_specification [SUPERUSER]
user_specification:
'user_name' [IDENTIFIED BY [PASSWORD] 'password']
```

Note:

The CREATE USER command can be used to create a palo user. When using this command, the user is required to have administrator privilege. SUPERUSER is used to specify that the user to be created is a superuser.

Example:

1. Create a user without password and with user name called jack

   CREATE USER 'jack'

2. Create a user with a password, whose name is jack, and password is specified as 123456

   CREATE USER 'jack' IDENTIFIED BY '123456'

3. To avoid passing plaintext, use case 2 can also be created by the following way

   CREATE USER 'jack' IDENTIFIED BY PASSWORD '*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9'

The encrypted content can be obtained through PASSWORD (), for example:

```
SELECT PASSWORD('123456')
```

4. Create a Super User' jack’

   CREATE USER 'jack' SUPERUSER

Drop user

This statement is used to delete a user, and the administrator privilege is required.

Syntax:

```
DROP USER 'user_name'
```

Example:

Delete the user jack

```
DROP USER 'jack'
```

Alter user

This statement is used to modify the relevant attributes of the user and the resources allocated to the user.

Syntax:

```
 ALTER USER user alter_user_clause_list
alter_user_clause_list:
alter_user_clause [, alter_user_clause] ...
alter_user_clause:
MODIFY RESOURCE resource value | MODIFY PROPERTY property value
resource:
CPU_SHARE
property:
MAX_USER_CONNECTIONS
```

Example:

1. Modify user jack's CPU_SHARE to 1000

   ALTER USER jack MODIFY RESOURCE CPU_SHARE 1000

2. Modify the maximum connection number of user jack to 1000

   ALTER USER jack MODIFY PROPERTY MAX_USER_CONNECTIONS 1000

Alter quota

This statement is used to modify the allocation of different groups of resources for a user.

Syntax:

```
ALTER QUOTA FOR user_name MODIFY group_name value
```

Example:

Modify the weight of the normal group of system users

```
ALTER QUOTA FOR system MODIFY normal 400;
```

Grant

This statement is used to authorize specific privileges of a database to specific users. The caller must be an administrator. The privilege currently includes READ_ONLY and READ_WRITE only. If ALL is specified, all privileges are authorized to the user.

Syntax:

```
 GRANT privilege_list ON db_name TO 'user_name'
privilege_list:
privilege [, privilege] ...
privilege:
READ_ONLY | READ_WRITE | ALL
```

Example:

1. Grant the user write privilege to the jack database testDb

GRANT READ_ONLY ON testDb to 'jack';

2. Grant all privileges to the user jack database testDb

   GRANT ALL ON testDb to 'jack';

Set password

This statement is used to modify a user's login password. If the [FOR 'user_name'] field does not exist, the password of the current user should be modified. PASSWORD () is a plaintext password. However, if the string is used directly, the encrypted password needs to be transferred. If you modify the password of other users, you need administrator privileges.

Syntax:

```
SET PASSWORD [FOR 'user_name'] = [PASSWORD('plain password')]|['hashed password']
```

Example:

1. Modify the password of the current user to 123456

   SET PASSWORD = PASSWORD('123456')
   SET PASSWORD = '*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9'

2. Modify the password of the user jack to 123456

   SET PASSWORD FOR 'jack' = PASSWORD('123456')
   SET PASSWORD FOR 'jack' = '*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9'

Cluster Management

Alter system

This statement is used to operate nodes within a cluster. (For administrators only!)

Syntax:

```
 1. Add nodes
   ALTER SYSTEM ADD BACKEND "host:heartbeat_port"[,"host:heartbeat_port"...];
2. Delete nodes
   ALTER SYSTEM DROP BACKEND "host:heartbeat_port"[,"host:heartbeat_port"...];
3. Node downline
   ALTER SYSTEM DECOMMISSION BACKEND "host:heartbeat_port"[,"host:heartbeat_port"...];
4. Add Brokers
   ALTER SYSTEM ADD BROKER broker_name "host:port"[,"host:port"...];
5. Reduce Brokers
   ALTER SYSTEM DROP BROKER broker_name "host:port"[,"host:port"...];
6. Delete all Brokers
   ALTER SYSTEM DROP ALL BROKER broker_name
```

Note:

- Host may be a host name or an ip address.
- Heartbeat_port is the heartbeat port of this node.
- Adding and deleting a node are a synchronous operation. These two operations do not consider the existing data on the node, and the node is directly deleted from the metadata. Please use it carefully.
- Node downline operation is used for safe downline nodes. This operation is asynchronous. If the operation is done successfully, the node is deleted from the metadata. If failed, the downline operation is not completed.

- You can manually cancel the node downline operation. For detailed information, see CANCEL ALTER SYSTEM.

Example:

1. Add a node

   ALTER SYSTEM ADD BACKEND "host:9850";

2. Delete two nodes

   ALTER SYSTEM DROP BACKEND "host1:9850", "host2:9850";

3. Enable two nodes to be downline

   ALTER SYSTEM DECOMMISSION BACKEND "host1:9850", "host2:9850";

4. Add two Hdfs Brokers

   ALTER SYSTEM ADD BROKER hdfs "host1:9850", "host2:9850";

Cancel alter system

This statement is used to revoke the downline operation of a node. (For administrators only!)

Syntax:

```
 CANCEL ALTER SYSTEM DECOMMISSION BACKEND "host:heartbeat_port"[,"host:heartbeat_port"...];
```

Example:

1. Cancel the downline operation of the two nodes

   CANCEL ALTER SYSTEM DECOMMISSION BACKEND "host1:9850", "host2:9850";

Create cluster

This statement is used to create a logical cluster in need of administrator privilege. If multi-tenancy is not used, create a cluster named default_cluster. Otherwise, create a cluster with a custom name.

Syntax:

```
 CREATE CLUSTER [IF NOT EXISTS] cluster_name
PROPERTIES ("key"="value", ...)
IDENTIFIED BY 'password'
```

PROPERTIES

It specifies the properties of the logical cluster. PROERTIES ("instance_num" = "3"). Where the instance_num is the number of logical cluster nodes.

identified by ‘password’

Each logical cluster contains a superuser whose password must be specified when creating the logical cluster.

Example:

1. Create a logical test_cluster with 3 be nodes and specify its superuser account password.

   CREATE CLUSTER test_cluster PROPERTIES("instance_num"="3") IDENTIFIED BY 'test';

2. Create a logical cluster default_cluster with 3 be nodes (without use of multi-tenant) and specify its superuser account password

   CREATE CLUSTER default_cluster PROPERTIES("instance_num"="3") IDENTIFIED BY 'test';

Alter cluster

This statement is used to update the logical cluster. Administrator rights are required.

Syntax:

```
ALTER CLUSTER cluster_name PROPERTIES ("key"="value", ...);
```

PROPERTIES

Capacity reduction and expansion. According to the existing number of bes for the cluster, the large capacity is obtained by expansion and the small capacity is obtained by reduction. Capacity expansion is synchronous and capacity reduction is asynchronous. Whether the capacity reduction is completed can be known through the status of backend. PROERTIES ("instance_num" = "3"). Where the instance_num is the number of logical cluster nodes.

Example:

1. Reduce the capacity and reduce the number of bes in logical cluster test_cluster containing 3 be to 2

   ALTER CLUSTER test_cluster PROPERTIES ("instance_num"="2");

2. Expand the capacity and increase the number of bes in logical cluster test_cluster containing 3 bes to 4

   ALTER CLUSTER test_cluster PROPERTIES ("instance_num"="4");

Drop cluster

This statement is used to delete a logical cluster. To successfully delete a logical cluster, you need to take administrator rights to delete the db in the cluster first, and you need administrator rights.

Syntax:

```
DROP CLUSTER [IF EXISTS] cluster_name;
```

Whitelist Management

Whitelist Format

The white list has 3 formats:

1. ip: 127.0.0.1

2. Combination of ip and * (equivalent to mask): 127.1.*.* or 127.0.1.*

3. hostName: hostname.beijing

Add a whitelist

Multiple whitelists are separated with commas.

```
ALTER USER user_name ADD WHITELIST  "ip1, ip2";
```

Delete the whitelist

```
ALTER USER jack DELETE WHITELIST  "ip1, ip2, ip3"
```

Display the whitelist

```
SHOW WHITELIST
```

Privileges

1. When the whitelist is not added, all ips can be accessed by default.

Once the whitelist is added, only the ip allowed by the whitelist can be accessed.

2. The superuser can set the whitelist of ordinary users.

Ordinary users with write privilege can only modify their own whitelist.

Built-in Function

Mathematical Functions

abs(double a)

Feature: Returns the absolute value of a parameter

Return type: Double type

Instructions for Use: Use this function to ensure that the return value of the function is an integer.

acos (double a)

Feature: Returns the inverse cosine of a parameter

Return type: Double type

asin(double a)

Feature: Returns the arcsine value of the parameter

Return type: Double type

atan(double a)

Feature: Returns the arctangent of a parameter

Return type: Double type

bin(bigint a)

Feature: Returns the binary representation of an integer (i.e., the sequence of 0 and 1)

Return type: String type

```
 mysql> select bin(10);
+---------+
| bin(10) |
+---------+
| 1010    |
+---------+
1 row in set (0.01 sec)
```

ceil(double a)

ceiling(double a)

dceil(double a)

Feature: Returns the smallest integer greater than or equal to this parameter

Return type: int type

conv(bigint num, int from_base, int to_base)

conv(string num,int from_base, int to_base)

Feature: It is a positional number system conversion function that returns the string form of an integer in a specific binary system. The input parameter may be an integer string. If you want to convert the return value of a function to an integer, you can use the CAST function.

Return type: string type

Example:

```
 mysql> select conv(64,10,8);
+-----------------+
| conv(64, 10, 8) |
+-----------------+
| 100             |
+-----------------+
1 row in set (0.01 sec)

mysql> select cast(conv('fe', 16, 10) as int) as "transform_string_to_int";
+-------------------------+
| transform_string_to_int |
+-------------------------+
|                     254 |
+-------------------------+
1 row in set (0.00 sec)
```

cos(double a)

Feature: It returns the cosine of a parameter

Return type: double type

degrees(double a)

Feature: It converts an radians into an angle.

Return type: double type

e()

Feature: It returns the mathematical constant e

Return type: double type

exp(double a)

dexp(double a)

Feature: It returns the e to the ath power (i.e., ea)

Return type: double type

floor(double a)

dfloor(double a)

Feature: It returns the largest integer less than or equal to this parameter

Return type: int type

fmod(double a, double b)

fmod(float a, float b)

Feature: It returns the remainder of a divided by b. It is equivalent to % arithmetic operator

Return type: float or double type

Example:

```
 mysql> select fmod(10,3);
+-----------------+
| fmod(10.0, 3.0) |
+-----------------+
|               1 |
+-----------------+
1 row in set (0.01 sec)

mysql> select fmod(5.5,2);
+----------------+
| fmod(5.5, 2.0) |
+----------------+
|            1.5 |
+----------------+
1 row in set (0.01 sec)
```

greatest(bigint a[, bigint b ...])

greatest(double a[, double b ...])

greatest(decimal(p,s) a[, decimal(p,s) b ...])

greatest(string a[, string b ...])

greatest(timestamp a[, timestamp b ...])

Feature: It returns the maximum value in the list

Return Type: Same as Parameter Type

hex(bigint a)

hex(string a)

Feature: It returns the hexadecimal representation of each character in an integer or a string.

Return type: string type

Example:

```
 mysql> select hex('abc');
+------------+
| hex('abc') |
+------------+
| 616263     |
+------------+
1 row in set (0.01 sec)

mysql> select unhex(616263);
+---------------+
| unhex(616263) |
+---------------+
| abc           |
+---------------+
1 row in set (0.01 sec)
```

least(bigint a[, bigint b ...])

least(double a[, double b ...])

least(decimal(p,s) a[, decimal(p,s) b ...])

least(string a[, string b ...])

least(timestamp a[, timestamp b ...])

Feature: It returns the minimum value in the list

Return Type: Same as Parameter Type

ln(double a)

dlog1(double a)

Feature: It returns the natural logarithmic form of the parameter

Return type: double type

log(double base, double a)

Feature: It returns the log logarithm with base as the base and a as the index.

Return type: double type

log10(double a)

dlog10(double a)

Feature: It returns the logarithm of log with 10 as the base and a as the index.

Return type: double type

log2(double a)

Feature: It returns the logarithm of log with 2 as the base and a as the index.

Return type: double type

mod(numeric_type a, same_type b)

Feature: It returns the remainder of a divided by b. Equivalent to % arithmetic operator.

Return Type: Same as Input Type

Example:

```
  mysql> select mod(10,3);
+------------+
| mod(10, 3) |
+------------+
|          1 |
+------------+
1 row in set (0.01 sec)

mysql> select mod(5.5,2);
+-------------+
| mod(5.5, 2) |
+-------------+
|         1.5 |
+-------------+
1 row in set (0.01 sec)
```

negative(int a)

negative(double a)

Feature: It inverts the sign bit of parameter a, and returns a positive value if the parameter is negative

Return type: It returns int or double based on the input parameter type

Instructions for use: If you need to ensure that all return values are negative, you can use the -abs(a) function.

pi()

Feature: It returns constant Pi

Return type: Double type

pmod(int a, int b)

pmod(double a, double b)

Feature: It is a positive remainder function

Return type: int type or double type (determined by input parameters)

positive(int a)

Feature: It returns the original value of the parameter. Even if the parameter is negative, the original value is still returned.

Return type: int type

Instructions for use: To ensure that all return values are positive, you can use the abs() function.

pow(double a, double p)

power(double a, double p)

Feature: It returns the a to the power of p.

Return type: double type

radians(double a)

Feature: It converts radian to angle

Return type: double type

rand()

rand(int seed)

random()

random(int seed)

Feature: It returns a random value between 0 and 1. Parameter is random seed.

Return type: double type

Instructions for use: The random sequence of each query is reset, and multiple calls to the rand function produce the same result. If you want to produce different results for each query, you can use different random seeds for each query. For example, select rand (unix _ timestamp ()) from ...

round(double a)

round(double a, int d)

Feature: It is a rounding function. If you take only one parameter, the function returns the integer nearest to the value. If there are 2 parameters, the second parameter is the number of digits rounded after the decimal point.

Return type: It returns bigint if the parameter is of floating-point type. If the second parameter is greater than 1, it returns a double type.

Example:

```
 mysql> select round(100.456, 2);
+-------------------+
| round(100.456, 2) |
+-------------------+
|            100.46 |
+-------------------+
1 row in set (0.02 sec)
```

sign(double a)

Feature: If a is an integer or 0, return 1; if a is negative, return -1

Return type: int type

sin(double a)

Feature: It returns the sine value of a

Return type: double type

sqrt(double a)

Feature: It returns the square root of a

Return type: double type

tan(double a)

Feature: It returns the tangent of a

Return type: double type

unhex(string a)

Feature: It converts a string from hexadecimal format to its original format

Return type: string type

Example:

```
 mysql> select hex('abc');
+------------+
| hex('abc') |
+------------+
| 616263     |
+------------+
1 row in set (0.01 sec)

mysql> select unhex(616263);
+---------------+
| unhex(616263) |
+---------------+
| abc           |
+---------------+
1 row in set (0.01 sec)
```

Bit operation function

bitand(integer_type a, same_type b)

Feature: Bitwise AND operation

Return type: Same as the input type

Example:

```
 mysql> select bitand(255, 32767); /* 0000000011111111 & 0111111111111111 */
+--------------------+
| bitand(255, 32767) |
+--------------------+
|                255 |
+--------------------+
1 row in set (0.01 sec)

mysql> select bitand(32767, 1); /* 0111111111111111 & 0000000000000001 */
+------------------+
| bitand(32767, 1) |
+------------------+
|                1 |
+------------------+
1 row in set (0.01 sec)

mysql> select bitand(32, 16); /* 00010000 & 00001000 */
+----------------+
| bitand(32, 16) |
+----------------+
|              0 |
+----------------+
1 row in set (0.01 sec)

mysql> select bitand(12,5); /* 00001100 & 00000101 */
+---------------+
| bitand(12, 5) |
+---------------+
|             4 |
+---------------+
1 row in set (0.01 sec)

mysql> select bitand(-1,15); /* 11111111 & 00001111 */
+----------------+
| bitand(-1, 15) |
+----------------+
|             15 |
+----------------+
1 row in set (0.01 sec)
```

bitnot(integer_type a)

Feature: Bitwise NOT operation

Return type: Same as the input type

Example:

```
 mysql> select bitnot(127); /* 01111111 -> 10000000 */
+-------------+
| bitnot(127) |
+-------------+
|        -128 |
+-------------+
1 row in set (0.01 sec)

mysql> select bitnot(16); /* 00010000 -> 11101111 */
+------------+
| bitnot(16) |
+------------+
|        -17 |
+------------+
1 row in set (0.01 sec)

mysql> select bitnot(0); /* 00000000 -> 11111111 */
+-----------+
| bitnot(0) |
+-----------+
|        -1 |
+-----------+
1 row in set (0.01 sec)

mysql> select bitnot(-128); /* 10000000 -> 01111111 */
+--------------+
| bitnot(-128) |
+--------------+
|          127 |
+--------------+
1 row in set (0.01 sec)
```

bitor(integer_type a, same_type b)

Feature: Bitwise OR operation

Return type: Same as the input type

Example:

```
 mysql> select bitor(1,4); /* 00000001 | 00000100 */
+-------------+
| bitor(1, 4) |
+-------------+
|           5 |
+-------------+
1 row in set (0.01 sec)

mysql> select bitor(16,48); /* 00001000 | 00011000 */
+---------------+
| bitor(16, 48) |
+---------------+
|            48 |
+---------------+
1 row in set (0.01 sec)

mysql> select bitor(0,7); /* 00000000 | 00000111 */
+-------------+
| bitor(0, 7) |
+-------------+
|           7 |
+-------------+
1 row in set (0.01 sec)
```

bitxor(integer_type a, same_type b)

Feature: Bitwise XOR operation

Return Type: Same as Input Type

Example:

```
mysql> select bitxor(0,15); /* 00000000 ^ 00001111 */
+---------------+
| bitxor(0, 15) |
+---------------+
|            15 |
+---------------+
1 row in set (0.01 sec)

mysql> select bitxor(7,7); /* 00000111 ^ 00000111 */
+--------------+
| bitxor(7, 7) |
+--------------+
|            0 |
+--------------+
1 row in set (0.01 sec)

mysql> select bitxor(8,4); /* 00001000 ^ 00000100 */
+--------------+
| bitxor(8, 4) |
+--------------+
|           12 |
+--------------+
1 row in set (0.01 sec)

mysql> select bitxor(3,7); /* 00000011 ^ 00000111 */
+--------------+
| bitxor(3, 7) |
+--------------+
|            4 |
+--------------+
1 row in set (0.01 sec)
```

Type Conversion

Functions

cast(expr as type)

The conversion function is usually used with other functions, and the displayed converts expression to the specified parameter type. Palo has strict data type definitions for parameter type of functions. For example, Palo does not automatically perform the bigtint-to-int conversion, or the remaining conversions that may lose precision or cause overflow. The user can use cast function to convert column values or literal constants into other types required by function parameters.

Example:

```
mysql> select concat('Here are the first ', cast(10 as string), ' results.');
+-------------------------------------------------------------------+
| concat('Here are the first ', CAST(10 AS CHARACTER), ' results.') |
+-------------------------------------------------------------------+
| Here are the first 10 results.                                    |
+-------------------------------------------------------------------+
1 row in set (0.01 sec)
```

Date and time functions

The time type supported by Palo is TIMESTAMP, including DATE and DATETIME. TIMESTAMP includes date and time. The date and time functions can extract a single field, such as hour(), minute (). Usually, the return value of such functions is an integer. The return value of a function that formats a date, such as date_add (), is of string type. The user can change the value of the time type by adding or subtracting the time interval. The time interval is usually the second parameter of date_add () and date_sub (). Palo supports the following date and time functions.

add_months(timestamp date, int months)

add_months(timestamp date, bigint months)

Feature: It returns the specified date plus the new date of months. It is the same as months_add ()

Return type: timestamp type

Example:

If this day of this month does not exist in the target month, the result is the last day of that month; if months in the parameter is negative, the result for the previous month is obtained.

```
 mysql> select now(), add_months(now(), 2);
+---------------------+---------------------+
| now()               | add_months(now(), 2)|
+---------------------+---------------------+
| 2016-05-31 10:47:00 | 2016-07-31 10:47:00 |
+---------------------+---------------------+
1 row in set (0.01 sec)

mysql> select now(), add_months(now(), 1);
+---------------------+---------------------+
| now()               | add_months(now(), 1)|
+---------------------+---------------------+
| 2016-05-31 10:47:14 | 2016-06-30 10:47:14 |
+---------------------+---------------------+
1 row in set (0.01 sec)

mysql> select now(), add_months(now(), -1);
+---------------------+----------------------+
| now()               | add_months(now(), -1)|
+---------------------+----------------------+
| 2016-05-31 10:47:31 | 2016-04-30 10:47:31  |
+---------------------+----------------------+
1 row in set (0.01 sec)
```

adddate(timestamp startdate, int days)

adddate(timestamp startdate, bigint days)

Feature: It adds the specified number of days to startdate.

Return type: timestamp type

Example:

```
 mysql> select adddate(date_column, 10) from big_table limit 1;
+-------------------------------+
|   adddate(date_column, 10)    |
+-------------------------------+
|          2014-01-11 00:00:00  |
+-------------------------------+
```

current_timestamp()

Feature: It is the same as now () function, which is able to obtain the current time

Return type: timestamp type

date_add(timestamp startdate, int days)

Feature: It adds the specified number of days to the TIMESTAMP value. The first parameter may be a string. If the string meets the format of TIMESTAMP data type, the string is automatically converted to TIMESTAMP type. The second parameter is the time interval.

Return type: timestamp type

date_format(timestamp day, string fmt)

Feature: It converts the date type into a string according to the format type, currently supports a string of up to 128 bytes, and if the length exceeds 128, returns NULL.

Return type: string type

Format has the following meaning:

```
  %a Abbreviated weekday name (Sun..Sat)
%b Abbreviated month name (Jan..Dec)
%c Month, numeric (0..12)
%D Day of the month with English suffix (0th, 1st, 2nd, 3rd, …)
%d Day of the month, numeric (00..31)
%e Day of the month, numeric (0..31)
%f Microseconds (000000..999999)
%H Hour (00..23)
%h Hour (01..12)
%I Hour (01..12)
%i Minutes, numeric (00..59)
%j Day of year (001..366)
%k Hour (0..23)
%l Hour (1..12)
%M Month name (January..December)
%m Month, numeric (00..12)
%p AM or PM
%r Time, 12-hour (hh:mm:ss followed by AM or PM)
%S Seconds (00..59)
%s Seconds (00..59)
%T Time, 24-hour (hh:mm:ss)
%U Week (00..53), where Sunday is the first day of the week; WEEK() mode 0
%u Week (00..53), where Monday is the first day of the week; WEEK() mode 1
%V Week (01..53), where Sunday is the first day of the week; WEEK() mode 2; used with %X
%v Week (01..53), where Monday is the first day of the week; WEEK() mode 3; used with %x
%W Weekday name (Sunday..Saturday)
%w Day of the week (0=Sunday..6=Saturday)
%X Year for the week where Sunday is the first day of the week, numeric, four digits; used with %V
%x Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v
%Y Year, numeric, four digits
%y Year, numeric (two digits)
%% A literal “%” character
%x x, for any “x” not listed above
```

Example:

```
 mysql> select date_format('2009-10-04 22:23:00', '%W %M %Y');
+------------------------------------------------+
| date_format('2009-10-04 22:23:00', '%W %M %Y') |
+------------------------------------------------+
| Sunday October 2009                            |
+------------------------------------------------+
1 row in set (0.01 sec)

mysql> select date_format('2007-10-04 22:23:00', '%H:%i:%s');
+------------------------------------------------+
| date_format('2007-10-04 22:23:00', '%H:%i:%s') |
+------------------------------------------------+
| 22:23:00                                       |
+------------------------------------------------+
1 row in set (0.01 sec)

mysql> select date_format('1900-10-04 22:23:00', '%D %y %a %d %m %b %j');
+------------------------------------------------------------+
| date_format('1900-10-04 22:23:00', '%D %y %a %d %m %b %j') |
+------------------------------------------------------------+
| 4th 00 Thu 04 10 Oct 277                                   |
+------------------------------------------------------------+
1 row in set (0.03 sec)

mysql> select date_format('1997-10-04 22:23:00', '%H %k %I %r %T %S %w');
+------------------------------------------------------------+
| date_format('1997-10-04 22:23:00', '%H %k %I %r %T %S %w') |
+------------------------------------------------------------+
| 22 22 10 10:23:00 PM 22:23:00 00 6                         |
+------------------------------------------------------------+
1 row in set (0.00 sec)
```

date_sub(timestamp startdate, int days)

Feature: It subtracts the specified number of days from the TIMESTAMP value. The first parameter may be a string. If the string meets the format of TIMESTAMP data type, the string is automatically converted to TIMESTAMP type. The second parameter is the time interval.

Return type: timestamp type

datediff(string enddate, string startdate)

Feature: It returns the difference between two dates

Return type: int type

day(string date)

dayofmonth(string date)

Feature: It returns the day field in a date

Return type: int type

Example:

```
  mysql> select dayofmonth('2013-01-21');
+-----------------------------------+
| dayofmonth('2013-01-21 00:00:00') |
+-----------------------------------+
|                                21 |
+-----------------------------------+
1 row in set (0.01 sec)
```

days_add(timestamp startdate, int days)

days_add(timestamp startdate, bigint days)

Feature: It is able to add the specified number of days to startdate, which is similar to the date_add function. The difference between these two functions is that the parameter of this function is of TIMESTAMP type, instead of string.

Return type: timestamp type

days_sub(timestamp startdate, int days)

days_sub(timestamp startdate, bigint days)

Feature: It subtracts the specified number of days from startdate, which is similar to the date_dub function. The difference between these two functions is that the parameter of this function is of TIMESTAMP type, instead of string.

Return type: timestamp type

extract(unit FROM timestamp)

Feature: It extracts the value of a specified unit of timestamp. The unit may be year, month, day, hour, minute or second.

Return type: int type

Example:

```
  mysql> select now() as right_now,
->   extract(year from now()) as this_year,
->   extract(month from now()) as this_month;
+---------------------+-----------+------------+
| right_now           | this_year | this_month |
+---------------------+-----------+------------+
| 2017-10-16 20:47:28 |      2017 |         10 |
+---------------------+-----------+------------+
1 row in set (0.01 sec)

mysql> select now() as right_now,
    ->   extract(day from now()) as this_day,
    ->   extract(hour from now()) as this_hour;
+---------------------+----------+-----------+
| right_now           | this_day | this_hour |
+---------------------+----------+-----------+
| 2017-10-16 20:47:34 |       16 |        20 |
+---------------------+----------+-----------+
1 row in set (0.01 sec)
```

from_unixtime(bigint unixtime[, string format])

Feature: It converts unix time (the number of seconds since January 1, 1970) to a date type in the appropriate format

Return type: String type

Instructions for use: The current date format is sensitive to the uppercase and lowercase letters. The user should distinguish the lowercase M (minute) from the uppercase M (month). The complete type of the date string is "yyyy-MM-dd HH:mm:ss.SSSSSS", or it may also contain some of the fields only.

Example:

```
 mysql> select from_unixtime(100000);
+-----------------------+
| from_unixtime(100000) |
+-----------------------+
| 1970-01-02 11:46:40   |
+-----------------------+
1 row in set (0.01 sec)

mysql> select from_unixtime(100000, 'yyyy-MM-dd');
+-------------------------------------+
| from_unixtime(100000, 'yyyy-MM-dd') |
+-------------------------------------+
| 1970-01-02                          |
+-------------------------------------+
1 row in set (0.00 sec)

mysql> select from_unixtime(1392394861, 'yyyy-MM-dd');
+-----------------------------------------+
| from_unixtime(1392394861, 'yyyy-MM-dd') |
+-----------------------------------------+
| 2014-02-15                              |
+-----------------------------------------+
1 row in set (0.00 sec)
```

Unix_timestamp () and from_unixtime () are often used together to convert the timestamp type into a string appearing in the specified format.

```
 mysql> select from_unixtime(unix_timestamp(now()), 'yyyy-MM-dd');
+----------------------------------------------------+
| from_unixtime(unix_timestamp(now()), 'yyyy-MM-dd') |
+----------------------------------------------------+
| 2014-01-01                                         |
+----------------------------------------------------+
```

hour(string date)

Feature: It returns the hour field of the date represented by the string

Return type: int type

hours_add(timestamp date, int hours)

hours_add(timestamp date, bigint hours)

Feature: It returns the specified date plus several hours.

Return type: timestamp

hours_sub(timestamp date, int hours)

hours_sub(timestamp date, bigint hours)

Feature: It returns the specified date minus several hours

Return type: timestamp

microseconds_add(timestamp date, int microseconds)

microseconds_add(timestamp date, bigint microseconds)

Feature: It returns the specified date plus several microseconds

Return type: timestamp

microseconds_sub(timestamp date, int microseconds)

microseconds_sub(timestamp date, bigint microseconds)

Feature: It returns the specified date minus several microseconds

Return type: timestamp

minute(string date)

Feature: It returns the minute field of the date represented by the string

Return type: int type

minutes_add(timestamp date, int minutes)

minutes_add(timestamp date, bigint minutes)

Feature: It returns the specified date plus several minutes

Return type: timestamp

minutes_sub(timestamp date, int minutes)

minutes_sub(timestamp date, bigint minutes)

Feature: It returns the specified date minus several minutes

Return type: timestamp

month(string date)

Feature: It returns the month field of the date expressed by the return string.

Return type: int type

months_add(timestamp date, int months)

months_add(timestamp date, bigint months)

Feature: It returns the specified date plus several months.

Return type: timestamp

months_sub(timestamp date, int months)

months_sub(timestamp date, bigint months)

Feature: It returns the specified date minus several months

Return type: timestamp

now()

Feature: It returns the current date and time (time zone of the 8th East Zone)

Return type: timestamp

second(string date)

Feature: It returns the second field of the date represented by the string

Return type: int type

seconds_add(timestamp date, int seconds)

seconds_add(timestamp date, bigint seconds)

Feature: It returns the specified date plus several seconds

Return type: timestamp

seconds_sub(timestamp date, int seconds)

seconds_sub(timestamp date, bigint seconds)

Feature: It returns the specified date minus several seconds

Return type: timestamp

subdate(timestamp startdate, int days)

subdate(timestamp startdate, bigint days)

Feature: It returns the startdate minus several days. It is similar to the date_sub () function, but the first parameter of this function is the exact TIMESTAMP, not a string that can be converted to TIMESTAMP type.

Return type: timestamp

str_to_date(string str, string format)

Feature: It converts str to timestamp type in the way specified by format, and if the conversion result is not correct, returns NULL. The supported format is consistent with date_format.

Return type: timestamp

to_date(timestamp)

Feature: It returns the date field of timestamp

Return type: string type

Example:

```
  mysql> select now() as right_now,
->   concat('The date today is ',to_date(now()),'.') as date_announcement;
+---------------------+-------------------------------+
| right_now           | date_announcement             |
+---------------------+-------------------------------+
| 2017-10-16 21:10:24 | The date today is 2017-10-16. |
+---------------------+-------------------------------+
1 row in set (0.01 sec)
```

unix_timestamp()

unix_timestamp(string datetime)

unix_timestamp(string datetime, string format)

unix_timestamp(timestamp datetime)

Feature: It returns the timestamp of the current time (number of seconds from January 1, 1970) or converts it from a specified date and time to a timestamp. The timestamp returned is that relative to Greenwich Time Zone.

Return type: bigint type

weeks_add(timestamp date, int weeks)

weeks_add(timestamp date, bigint weeks)

Feature: It returns the specified date plus several weeks

Return type: timestamp

weekofyear(timestamp date)

Feature: It is able to get the nth week of the year

Return type: int

weeks_add(timestamp date, int weeks)

weeks_add(timestamp date, bigint weeks)

Feature: It returns the specified date plus several weeks

Return type: timestamp

weeks_sub(timestamp date, int weeks)

weeks_sub(timestamp date, bigint weeks)

Feature: It returns the specified date minus several weeks

Return type: timestamp

quarter(timestamp date)

Feature: It returns the quarterly return type of the specified date: int

year(string date)

Feature: It returns the year field of the date represented by the string

Return type: int type

years_add(timestamp date, int years)

years_add(timestamp date, bigint years)

Feature: It returns the specified date plus several years

Return type: timestamp

years_sub(timestamp date, int years)

years_sub(timestamp date, bigint years)

Feature: It returns the specified date minus several years

Return type: timestamp

Conditional Functions

CASE a WHEN b THEN c [WHEN d THEN e]... [ELSE f] END

Feature: It compares the expression with several possible values, and when they match, returns the corresponding results.

Return type: The type of result returned after matching

Example:

```
  mysql> select case tiny_column when 1 then "tiny_column=1" when 2 then "tiny_column=2" end from small_table limit
2;
 +-------------------------------------------------------------------------------+
 | CASE`tiny_column` WHEN 1 THEN 'tiny_column=1' WHEN 2 THEN 'tiny_column=2' END |
 +-------------------------------------------------------------------------------+
 | tiny_column=1                                                                 |
 | tiny_column=2                                                                 |
 +-------------------------------------------------------------------------------+
 2 rows in set (0.02 sec)
```

if(boolean condition, type ifTrue, type ifFalseOrNull)

Feature: It is able to test an expression and returns the corresponding result depending on whether the result is true or false

Return type: type of ifTrue expression result

Example

```
  mysql> select if(tiny_column = 1, "true", "false") from small_table limit 1;
 +----------------------------------------+
 | if(`tiny_column` = 1, 'true', 'false') |
 +----------------------------------------+
 | true                                   |
 +----------------------------------------+
 1 row in set (0.03 sec)
```

ifnull(type a, type isNull)

Feature: It is able to test an expression, returns the second parameter if the expression is NULL; otherwise, it returns the first parameter.

Return type: Type of the first parameter

Example

```
  mysql> select ifnull(1,0);
 +--------------+
 | ifnull(1, 0) |
 +--------------+
 |            1 |
 +--------------+
 1 row in set (0.01 sec)

 mysql> select ifnull(null,10);
 +------------------+
 | ifnull(NULL, 10) |
 +------------------+
 |               10 |
 +------------------+
 1 row in set (0.01 sec)
```

nullif(expr1,expr2)

Feature: If the two parameters are equal, return NULL. Otherwise, return the value of the first parameter. It has the same effect as the following CASE WHEN.

```
 CASE
   WHEN expr1 = expr2 THEN NULL
   ELSE expr1
END
```

Return type: expr1 type or NULL

Example

```
  mysql> select nullif(1,1);
+--------------+
| nullif(1, 1) |
+--------------+
|         NULL |
+--------------+
1 row in set (0.00 sec)

mysql> select nullif(1,0);
+--------------+
| nullif(1, 0) |
+--------------+
|            1 |
+--------------+
1 row in set (0.01 sec)
```

String Processing Functions

ascii(string str)

Feature: It returns the ascii code corresponding to the first string of a string

Return type: int type

concat(string a, string b...)

Feature: It is able to connect multiple strings

Return type: string type

Instructions for use: concat () and concat_ws () combine multiple columns in a row into a new column, and group_concat () is an aggregate function which combines results from different rows into a new column.

concat_ws(string sep, string a, string b...)

Feature: It is able to connect the second parameter and the following parameters, and the connector is the first parameter.

Return type: string type

Example:

```
  mysql> select concat_ws('a', 'b', 'c', 'd');
+-------------------------------+
| concat_ws('a', 'b', 'c', 'd') |
+-------------------------------+
| bacad                         |
+-------------------------------+
1 row in set (0.01 sec)
```

find_in_set(string str, string strList)

Feature: It returns the location of the first str in strlist (counting from 1). StrList separates multiple strings with commas. If str is not found in strList, it returns 0.

Return type: int type

Example:

```
  mysql> select find_in_set("beijing", "tianji,beijing,shanghai");
+---------------------------------------------------+
| find_in_set('beijing', 'tianji,beijing,shanghai') |
+---------------------------------------------------+
|                                                 2 |
+---------------------------------------------------+
1 row in set (0.00 sec)
```

group_concat(string s [, string sep])

Feature: This function is an aggregate function similar to sum (), and the group_concat concatenates multiple rows of results in the result set to form a string. The second parameter is the connector between strings, which may be omitted. This function usually needs to be used with the group by statement.

Return type: string type

instr(string str, string substr)

Feature: It returns the position of substr first appearing in str, which counts from 1. If substr does not appear in str, it returns 0.

Return type: int type

Example:

```
  mysql> select instr('foo bar bletch', 'b');
+-----------------------------+
| instr('foo bar bletch', 'b') |
+-----------------------------+
|                           5 |
+-----------------------------+
1 row in set (0.01 sec)

mysql> select instr('foo bar bletch', 'z');
+-----------------------------+
| instr('foo bar bletch', 'z') |
+-----------------------------+
|                           0 |
+-----------------------------+
1 row in set (0.01 sec)
```

length(string a)

Feature: It returns the length of a string

Return type: int type

locate(string substr, string str[, int pos])

Feature: Ir returns the position of substr in str, which counts from 1. If the third parameter is specified, the position where substr appears is searched from the string where str starts with pos subscript.

Return type: int type

Example:

```
  mysql> select locate('bj', 'where is bj', 10);
+---------------------------------+
| locate('bj', 'where is bj', 10) |
+---------------------------------+
|                              10 |
+---------------------------------+
1 row in set (0.01 sec)

mysql> select locate('bj', 'where is bj', 11);
+---------------------------------+
| locate('bj', 'where is bj', 11) |
+---------------------------------+
|                               0 |
+---------------------------------+
1 row in set (0.01 sec)
```

lower(string a)

lcase(string a)

Feature: It converts all strings in the parameter to lowercase letter.

Return type: string type

lpad(string str, int len, string pad)

Feature: It returns a string with length len and counting from the initial letter in str. If len is longer than the str, pad characters are added before str until the length of the string reaches len. If len is shorter than the str, this function is equivalent to truncating str strings and returns a string with length equal to len.

Return type: string type

Example:

```
  mysql> select lpad("hello", 10, 'xy');
+-------------------------+
| lpad('hello', 10, 'xy') |
+-------------------------+
| xyxyxhello              |
+-------------------------+
1 row in set (0.01 sec)
```

ltrim(string a)

Feature: It is delete the spaces that appear continuously from the beginning of the parameter.

Return type: string type

regexp_extract(string subject, string pattern, int index)

Feature: It is able to perform regular matching of strings. Where the index is 0, it returns the entire matching string, where the index is 1,2, ..., it returns the first, second, ... part.

Return type: string type

Example:

```
 mysql> select regexp_extract('AbcdBCdefGHI','.*?([[:lower:]]+)',1);
+--------------------------------------------------------+
| regexp_extract('AbcdBCdefGHI', '.*?([[:lower:]]+)', 1) |
+--------------------------------------------------------+
| def                                                    |
+--------------------------------------------------------+
1 row in set (0.01 sec)

mysql> select regexp_extract('AbcdBCdefGHI','.*?([[:lower:]]+).*?',1);
+-----------------------------------------------------------+
| regexp_extract('AbcdBCdefGHI', '.*?([[:lower:]]+).*?', 1) |
+-----------------------------------------------------------+
| bcd                                                       |
+-----------------------------------------------------------+
1 row in set (0.01 sec)
```

regexp_replace(string initial, string pattern, string replacement)

Feature: It replaces the part of the initial string that matches the pattern.

Return type: string type

Example:

```
 mysql> select regexp_replace('aaabbbaaa','b+','xyz');
+------------------------------------------+
| regexp_replace('aaabbbaaa', 'b+', 'xyz') |
+------------------------------------------+
| aaaxyzaaa                                |
+------------------------------------------+
1 row in set (0.01 sec)

mysql> select regexp_replace('aaabbbaaa','(b+)','<\\1>');
+---------------------------------------------+
| regexp_replace('aaabbbaaa', '(b+)', '<\1>') |
+---------------------------------------------+
| aaa<bbb>aaa                                 |
+---------------------------------------------+
1 row in set (0.01 sec)

mysql> select regexp_replace('123-456-789','[^[:digit:]]','');
+---------------------------------------------------+
| regexp_replace('123-456-789', '[^[:digit:]]', '') |
+---------------------------------------------------+
| 123456789                                         |
+---------------------------------------------------+
1 row in set (0.01 sec)
```

repeat(string str, int n)

Feature: It returns the result of string str repeated for n times

Return type: string type

Example:

```
 mysql> select repeat("abc", 3);
+------------------+
| repeat('abc', 3) |
+------------------+
| abcabcabc        |
+------------------+
1 row in set (0.01 sec)
```

reverse(string a)

Feature: It inverts the string

Return type: string type

rpad(string str, int len, string pad)

Feature: It returns a string with length len and counting from the initial letter in str. If len is longer than the str, pad characters are added after str until the length of the string reaches len. If len is shorter than the str, this function is equivalent to truncating str strings and returns a string with length equal to len.

Return type: string type

Example:

```
 mysql> select rpad("hello", 10, 'xy');
+-------------------------+
| rpad('hello', 10, 'xy') |
+-------------------------+
| helloxyxyx              |
+-------------------------+
1 row in set (0.00 sec)
```

rtrim(string a)

Feature: It removes the spaces that appear continuously from the right part of the parameter.

Return type: string type

space(int n)

Feature: It returns a string of n spaces

Return type: string type

strleft(string a, int num_chars)

Feature: It returns the leftmost num_chars characters in a string.

Return type: string type

strright(string a, int num_chars)

Feature: It returns the rightmost num_chars characters in a string.

Return type: string type

substr(string a, int start [, int len])

substring(string a, int start[, int len])

Feature: The substring function is able to return part of the string which is described in the first parameter, and has length counting from start and equal to len. The initial letter has a subscript of 1.

Return type: string type

trim(string a)

Feature: It removes the spaces that appear continuously in the right part and the spaces that appear continuously in the left part of the parameter. This function has the same effect as using both ltrim () and rtrim ().

Return type: string type

upper(string a)

ucase(string a)

Feature: It converts all letters of a string to uppercase.

Return type: string type

Aggregation Function

AVG function

Feature: This aggregate function returns the average in the collection. The function has only one parameter, which can be a column of numeric type. The return value is a function of numeric, or the calculation result is a numeric expression. Rows containing NULL values are ignored. If the table is empty or the parameters of AVG are NULL, the function returns NULL. When the query specifies the use of the GROUP BY clause, 1 result is returned for the value of each group by.

Return type: Double type

COUNT Function

Feature: The aggregate function returns the number of rows that meet the requirements, or the number of non-NULL rows. COUNT(*) counts number of rows that contain NULL value. COUNT(column_name) only counts number of rows with non-NULL value. The user can use the COUNT function and the DISTINCT operator at the same time. count(distinct col_name ...) first de-duplicates the data, and then counts the number of occurrences of the combination of multiple columns.

Return type: int type

Example:

```
 mysql> select count(distinct tiny_column, short_column) from small_table;
+----------------------------------------------+
| count(DISTINCT `tiny_column`, `short_column`) |
+----------------------------------------------+
|                                            2 |
+----------------------------------------------+
1 row in set (0.08 sec)
```

MAX Function

Feature: This aggregate function returns the maximum value in the collection. This function has the function opposite to the min function. The function has only one parameter, which can be a column of numeric type. The return value is a function of numeric, or the calculation result is a numeric expression. Rows containing NULL values are ignored. If the table is empty or the parameters of MAX are NULL, the function returns NULL. When the query specifies the use of the GROUP BY clause, 1 result is returned for the value of each group by.

Return type: It is the same type as the input parameter.

MIN Function

Feature: This aggregate function returns the minimum value in the collection. This function is opposite to max function. The function has only one parameter, which can be a column of numeric type. The return value is a function of numeric, or the calculation result is a numeric expression. Rows containing NULL values are ignored. If the table is empty or the parameters of MIN are NULL, the function returns NULL. When the query specifies the use of the GROUP BY clause, 1 result is returned for the value of each group by.

Return type: It is the same type as the input parameter.

SUM Function

Feature: This aggregate function returns the sum of all values in the collection. The function has only one parameter, which can be a column of numeric type. The return value is a function of numeric, or the calculation result is a numeric expression. Rows containing NULL values are ignored. If the table is empty or the parameters of MIN are NULL, the function returns NULL. When the query specifies the use of the GROUP BY clause, 1 result is returned for the value of each group by.

Return Type: BIGINT if argument is integer, double if argument is floating point

GROUP_CONCAT Function

Feature: The aggregate function returns a string, which is a new string formed by connecting all strings in the collection. If the user specifies a separator, the separator is used to connect strings in two adjacent rows.

Return type: string type

Instructions for use: By default, this function returns a string that covers all result sets. When the query specifies the use of the group by clause, 1 result is returned for the value of each group by. Concat () and concat_ws () combine multiple columns in a row into a new column, and group_concat () is an aggregate function that combines results for different rows into a new column.

Variance Function

Syntax:

```
VARIANCE | VAR[IANCE]_SAMP | VAR[IANCE]_POP
```

Feature: This type of aggregate function returns the variance for a group of numbers. This is a mathematical attribute that represents the distance between the value and the average. It acts on numeric types. VARIANCE_SAMP () and VARIANCE_POP () are used to calculate the sample variance and population variance respectively, and VARIANCE () is the alias of VARIANCE_SAMP (). VAR_SAMP () and VAR_POP () are respectively VARIANCE_SAMP () and VARIANCE_POP () are aliases.

Return type: double type

Standard Deviation Function

Syntax:

```
STDDEV | STDDEV_SAMP | STDDEV_POP
```

Feature: This type of aggregate function returns the standard deviation for a set of numbers. It acts on numeric types. STDDEV_POP () and STDDEV_SAMP () are used to calculate the population standard deviation and sample standard deviation respectively. STDDEV () is an alias for STDDEV_SAMP ().

Return type: double type

Json Parsing Functions

Palo currently supports 3 json parsing functions

- get_json_int (string, string)
- get_json_string (string, string)
- get_json_double (string, string)

The first parameter is the json string and the second parameter is the path within json

Example:

```
  mysql> select get_json_int('{"col1":100, "col2":"string", "col3":1.5}', "$.col1");
+-------------------------------------------------------------------+
| get_json_int('{"col1":100, "col2":"string", "col3":1.5}', '$.col1') |
+-------------------------------------------------------------------+
|                                                               100 |
+-------------------------------------------------------------------+
1 row in set (0.01 sec)

mysql> select get_json_string('{"col1":100, "col2":"string", "col3":1.5}', "$.col2");
+---------------------------------------------------------------------+
| get_json_string('{"col1":100, "col2":"string", "col3":1.5}', '$.col2') |
+---------------------------------------------------------------------+
| string                                                              |
+---------------------------------------------------------------------+
1 row in set (0.01 sec)

mysql> select get_json_double('{"col1":100, "col2":"string", "col3":1.5}', "$.col3");
+---------------------------------------------------------------------+
| get_json_double('{"col1":100, "col2":"string", "col3":1.5}', '$.col3') |
+---------------------------------------------------------------------+
|                                                                 1.5 |
+---------------------------------------------------------------------+
1 row in set (0.01 sec)
```

HLL Function

HLL is an engineering implementation based on the HyperLogLog algorithm. It is used to save the intermediate results of the HyperLogLog calculation process. It can only be used as the value column type of the table and continuously reduce the data size through aggregation to achieve the purpose of speeding up the query. Based on an estimation result, the error is about 1%, and HLL column is generated through other columns or data in the imported data. When importing, hll_hash function is used to specify which column in the data is used to generate hll column, which is often used to replace count distinct and quickly calculate uv in business by combining rollup, etc.

HLL_UNION_AGG(hll)

This function is an aggregate function and is used to calculate the cardinality estimate of all data that meet the conditions.

HLL_CARDINALITY(hll)

This function is used to calculate the cardinality estimate for a single hll column

HLL_HASH(column_name)

When HLL column type is generated for insertion or import, refer to the relevant instructions for the use of import.

Example: (for illustration only)

1. First create a table with hll column:

```
  create table test(
 time date,
 id int,
 name char(10),
 province char(10),
 os char(1),
 set1 hll hll_union,
 set2 hll hll_union)
 distributed by hash(id) buckets 32;
```

2. Import data. See relevant help mini load for the import method.

```
 (1) Generate "hll" columns using columns from a table.
   curl --location-trusted -uname:password -T data http://host/api/test_db/test/_load?
label=load_1\&hll=set1,id:set2,name
(2) Generate a "hll" column from a column in the data.
   curl --location-trusted -uname:password -T data http://host/api/test_db/test/_load?
label=load_1\&hll=set1,cuid:set2,os\&columns=time,id,name,province,sex,cuid,os
```

3. There are three common ways to aggregate data: if you do not aggregate and directly query the base table, the speed may be the same as that of using ndv directly.

```
 (1) Create a rollup to make the "hll" columns aggregate.
   alter table test add rollup test_rollup(date, set1);
(2) Create another table that calculates UV and insert data)
   create table test_uv(
   time date,
   uv_set hll hll_union)
   distributed by hash(id) buckets 32;
   insert into test_uv select date, set1 from test;
(3) Create another table that calculates UV and insert data, and then insert and generate "hll" columns based on test
other "non-hll" columns by "hll_hash"
   create table test_uv(
   time date,
   id_set hll hll_union)
   distributed by hash(id) buckets 32;
   insert into test_uv select date, hll_hash(id) from test;
```

4. Query: It is not allowed that its original value in the hll column is queried directly but through matching functions.

```
 (1) Get total "uv"
   select HLL_UNION_AGG(uv_set) from test_uv;
(2) Get "uv" every day
   select HLL_CARDINALITY(uv_set) from test_uv;
```

Analysis Functions (window functions)

Introduction of Analysis Functions

Analytic function is a kind of special built-in function. Similar to the aggregate function, the analytic function is used to calculate a data value for multiple input rows. The difference is that the analytic function processes the input data in a specific window, instead of group calculation according to group by. The data in each window can be sorted and grouped by the over () clause. The analytic function is to calculate a separate value for each row in the result set, but not to calculate a value for each group by group. This flexible way allows users to add additional columns to the select clause, giving users more opportunities to reorganize and filter the result set. The analytic functions can only appear in the select list and the outermost order by clause. During the query process, the analytic function takes effect at the end, i.e.c, it is executed after the join, where, and group by operations are completed. Analytic functions are often used in the finance and scientific computation fields to analyze trends, calculate outliers, and perform bucketing analysis on a large amount of data.

Syntax of analytic function:

```
 function(args) OVER(partition_by_clause order_by_clause [window_clause])
partition_by_clause ::= PARTITION BY expr [, expr ...]
order_by_clause ::= ORDER BY expr [ASC | DESC] [, expr [ASC | DESC] ...]
```

window_clause: See Window Clause later.

Function

The currently supported function includes AVG (), COUNT (), DENSITY _ RANK (), FIRST _ VALUE (), LAG (), LAST _ VALUE (), LEAD (), MAX (), MIN (), RANK (), ROW _ NUMBER (), and SUM ().

PARTITION BY Clause

Partition By clause is similar to Group By clause. It groups input rows into one or more specified columns, and rows with the same value are grouped.

ORDER BY Clause

The Order By clause is basically the same as the outer Order By clause. It defines the order in which the input rows are arranged, and if Partition By is specified, Order By defines the order within each Partition group. The only difference with the outer Order By is that Order By n(n is a positive integer) in the OVER clause is equivalent to doing nothing, while Order By n in the outer layer means sorting by column n.

Example:

This example shows addition of an id column to the select list, and its values are 1, 2, 3, and so on, and sorted by the date_and_time column in the events table.

```
 SELECT
row_number() OVER (ORDER BY date_and_time) AS id,
c1, c2, c3, c4
FROM events;
```

Window Clause

The Window clause is used to specify an operation range for the analytic function. By taking the current row as the target row, several rows before and after the current row are taken as the operation objects of the analytic function. The methods supported by the Window clause are: avg (), COUNT (), FIRST _ VALUE (), LAST _ VALUE (), and SUM (). For MAX () and MIN (), the window clause can specify the starting range UNBOUNDED PRECEDING.

Syntax:

```
 ROWS BETWEEN [ { m | UNBOUNDED } PRECEDING | CURRENT ROW] [ AND [CURRENT ROW | { UNBOUNDED | n }
FOLLOWING] ]
```

Example:

Supposing that you have the following stock data, the stock code is JDR, and the closing price is the daily closing price.

```
 create table stock_ticker (stock_symbol string, closing_price decimal(8,2), closing_date timestamp);
...load some data...
select * from stock_ticker order by stock_symbol, closing_date
 | stock_symbol | closing_price | closing_date |
 |--------------|---------------|---------------------|
 | JDR | 12.86 | 2014-10-02 00:00:00 |
 | JDR | 12.89 | 2014-10-03 00:00:00 |
 | JDR | 12.94 | 2014-10-04 00:00:00 |
 | JDR | 12.55 | 2014-10-05 00:00:00 |
 | JDR | 14.03 | 2014-10-06 00:00:00 |
 | JDR | 14.75 | 2014-10-07 00:00:00 |
 | JDR | 13.98 | 2014-10-08 00:00:00 |
```

This query uses the analytic function to generate the column moving_average, whose value is the average share price for 3 days, i.e., the average price for the previous day, the current day and the next day. On the first day, no value for the previous day is available, and on the last day, no value for the next day available, so these two rows only calculate the average value for the two days. Partition By doesn't work here, because all the data are JDR data. However, if there is other stock information, Partition By ensures that the analytic function value takes effects within this Partition.

```
 select stock_symbol, closing_date, closing_price,
avg(closing_price) over (partition by stock_symbol order by closing_date
rows between 1 preceding and 1 following) as moving_average
from stock_ticker;
 | stock_symbol | closing_date | closing_price | moving_average |
 |--------------|---------------------|---------------|----------------|
 | JDR | 2014-10-02 00:00:00 | 12.86 | 12.87 |
 | JDR | 2014-10-03 00:00:00 | 12.89 | 12.89 |
 | JDR | 2014-10-04 00:00:00 | 12.94 | 12.79 |
 | JDR | 2014-10-05 00:00:00 | 12.55 | 13.17 |
 | JDR | 2014-10-06 00:00:00 | 14.03 | 13.77 |
 | JDR | 2014-10-07 00:00:00 | 14.75 | 14.25 |
 | JDR | 2014-10-08 00:00:00 | 13.98 | 14.36 |
```

Examples of Functions Use

This section describes methods that can be used as analytic function in Palo.

AVG()

Syntax:

AVG([DISTINCT ALL

Example:

Calculate the x average for the current row, and the ones before and after the current row.

```
select x, property,
avg(x) over
(
partition by property
order by x
rows between 1 preceding and 1 following
) as 'moving average'
from int_t where property in ('odd','even');
 | x | property | moving average |
 |----|----------|----------------|
 | 2 | even | 3 |
 | 4 | even | 4 |
 | 6 | even | 6 |
 | 8 | even | 8 |
 | 10 | even | 9 |
 | 1 | odd | 2 |
 | 3 | odd | 3 |
 | 5 | odd | 5 |
 | 7 | odd | 7 |
 | 9 | odd | 8 |
```

COUNT()

Syntax:

```
COUNT([DISTINCT | ALL] expression) [OVER (analytic_clause)]
```

Example:

Count the number of times the x appears from the current row to the first row.

```
select x, property,
count(x) over
(
partition by property
order by x
rows between unbounded preceding and current row
) as 'cumulative total'
from int_t where property in ('odd','even');
 | x | property | cumulative count |
 |----|----------|------------------|
 | 2 | even | 1 |
 | 4 | even | 2 |
 | 6 | even | 3 |
 | 8 | even | 4 |
 | 10 | even | 5 |
 | 1 | odd | 1 |
 | 3 | odd | 2 |
 | 5 | odd | 3 |
 | 7 | odd | 4 |
 | 9 | odd | 5 |
```

DENSE_RANK()

The DENSE_RANK () function is used to indicate ranking. Unlike rank (), DENSE_RANK () does not miss a number. For example, if 1 appears twice in parallel, the third number of DENSE_RANK () is still 2, and the third number of RANK () is 3.

Syntax:

```
DENSE_RANK() OVER(partition_by_clause order_by_clause)
```

Example:

The following example shows the ranking of the x column by grouping property columns:

```
 select x, y, dense_rank() over(partition by x order by y) as rank from int_t;
| x | y | rank |
|----|------|----------|
| 1 | 1 | 1 |
| 1 | 2 | 2 |
| 1 | 2 | 2 |
| 2 | 1 | 1 |
| 2 | 2 | 2 |
| 2 | 3 | 3 |
| 3 | 1 | 1 |
| 3 | 1 | 1 |
| 3 | 2 | 2 |
```

FIRST_VALUE()

FIRST_VALUE () returns the first value within the window range.

Syntax:

```
FIRST_VALUE(expr) OVER(partition_by_clause order_by_clause [window_clause])
```

Example:

We have the following data

```
 select name, country, greeting from mail_merge;
| name | country | greeting |
|---------|---------|--------------|
| Pete | USA | Hello |
| John | USA | Hi |
| Boris | Germany | Guten tag |
| Michael | Germany | Guten morgen |
| Bjorn | Sweden | Hej |
| Mats | Sweden | Tja |
```

Use FIRST_VALUE () to return the value of the first greeting in each group according to the country group:

```
 select country, name,
first_value(greeting)
over (partition by country order by name, greeting) as greeting from mail_merge;
| country | name | greeting |
|---------|---------|-----------|
| Germany | Boris | Guten tag |
| Germany | Michael | Guten tag |
| Sweden | Bjorn | Hej |
| Sweden | Mats | Hej |
| USA | John | Hi |
| USA | Pete | Hi |
```

LAG()

The LAG () method is used to calculate the value for the several rows ahead of the current row.

Syntax:

```
LAG (expr, offset, default) OVER (partition_by_clause order_by_clause)
```

Example:

Calculate the closing price of the previous day.

```
select stock_symbol, closing_date, closing_price,
lag(closing_price,1, 0) over (partition by stock_symbol order by closing_date) as "yesterday closing"
from stock_ticker
order by closing_date;
| stock_symbol | closing_date | closing_price | yesterday closing |
|--------------|---------------------|---------------|-------------------|
| JDR | 2014-09-13 00:00:00 | 12.86 | 0 |
| JDR | 2014-09-14 00:00:00 | 12.89 | 12.86 |
| JDR | 2014-09-15 00:00:00 | 12.94 | 12.89 |
| JDR | 2014-09-16 00:00:00 | 12.55 | 12.94 |
| JDR | 2014-09-17 00:00:00 | 14.03 | 12.55 |
| JDR | 2014-09-18 00:00:00 | 14.75 | 14.03 |
| JDR | 2014-09-19 00:00:00 | 13.98 | 14.75
```

LAST_VALUE()

LAST_VALUE () returns the last value within the window range. It is on the contrary to FIRST_VALUE ().

Syntax:

```
LAST_VALUE(expr) OVER(partition_by_clause order_by_clause [window_clause])
```

Use the data in the FIRST_VALUE () example:

```
select country, name,
last_value(greeting)
over (partition by country order by name, greeting) as greeting
from mail_merge;
| country | name | greeting |
|---------|---------|--------------|
| Germany | Boris | Guten morgen |
| Germany | Michael | Guten morgen |
| Sweden | Bjorn | Tja |
| Sweden | Mats | Tja |
| USA | John | Hello |
| USA | Pete | Hello
```

LEAD()

The LEAD () method is used to calculate the value for several rows after the current row.

Syntax:

```
LEAD (expr, offset, default]) OVER (partition_by_clause order_by_clause)
```

Example:

Calculate the trend of the closing price for the next day compared with the closing price for the current day, i.e., whether the closing price for the next day is higher or lower than the closing price for the current day.

```
select stock_symbol, closing_date, closing_price,
case
(lead(closing_price,1, 0)
over (partition by stock_symbol order by closing_date)-closing_price) > 0
when true then "higher"
when false then "flat or lower"
end as "trending"
from stock_ticker
order by closing_date;
| stock_symbol | closing_date | closing_price | trending |
|--------------|---------------------|---------------|---------------|
| JDR | 2014-09-13 00:00:00 | 12.86 | higher |
| JDR | 2014-09-14 00:00:00 | 12.89 | higher |
| JDR | 2014-09-15 00:00:00 | 12.94 | flat or lower |
| JDR | 2014-09-16 00:00:00 | 12.55 | higher |
| JDR | 2014-09-17 00:00:00 | 14.03 | higher |
| JDR | 2014-09-18 00:00:00 | 14.75 | flat or lower |
| JDR | 2014-09-19 00:00:00 | 13.98 | flat or lower |
```

MAX()

Syntax:

```
MAX([DISTINCT | ALL] expression) [OVER (analytic_clause)]
```

Example:

Calculate the maximum value from the first row to the row following the current row

```
select x, property,
max(x) over
(
order by property, x
rows between unbounded preceding and 1 following
) as 'local maximum'
from int_t where property in ('prime','square');
| x | property | local maximum |
|---|----------|---------------|
| 2 | prime | 3 |
| 3 | prime | 5 |
| 5 | prime | 7 |
| 7 | prime | 7 |
| 1 | square | 7 |
| 4 | square | 9 |
| 9 | square | 9 |
```

MIN()

Syntax:

```
MIN([DISTINCT | ALL] expression) [OVER (analytic_clause)]
```

Example:

Calculate the minimum value from the first row to the row following the current row

```
  select x, property,
min(x) over
(
order by property, x desc
rows between unbounded preceding and 1 following
) as 'local minimum'
from int_t where property in ('prime','square');
| x | property | local minimum |
|---|----------|---------------|
| 7 | prime | 5 |
| 5 | prime | 3 |
| 3 | prime | 2 |
| 2 | prime | 2 |
| 9 | square | 2 |
| 4 | square | 1 |
| 1 | square | 1 |
```

RANK()

The RANK() function is used to indicate ranking. Unlike DENSE_RANK(), RANK() may miss a number. For example, if 1 appears twice in parallel, the third number of RANK() is 3 instead of 2.

Syntax:

```
  RANK() OVER(partition_by_clause order_by_clause)
```

Example:

Rank according to column x

```
  select x, y, rank() over(partition by x order by y) as rank from int_t;
| x | y | rank |
|----|------|----------|
| 1 | 1 | 1 |
| 1 | 2 | 2 |
| 1 | 2 | 2 |
| 2 | 1 | 1 |
| 2 | 2 | 2 |
| 2 | 3 | 3 |
| 3 | 1 | 1 |
| 3 | 1 | 1 |
| 3 | 2 | 3 |
```

ROW_NUMBER()

It returns an integer that increases continuously from 1 for each row of each Partition. Unlike RANK () and DENSE_RANK (), the value returned by ROW_NUMBER () is not repeated or missed, but is continuously increasing.

Syntax:

```
  ROW_NUMBER() OVER(partition_by_clause order_by_clause)
```

Example:

```
  select x, y, row_number() over(partition by x order by y) as rank from int_t;
| x | y | rank |
|---|------|----------|
| 1 | 1 | 1 |
| 1 | 2 | 2 |
| 1 | 2 | 3 |
| 2 | 1 | 1 |
| 2 | 2 | 2 |
| 2 | 3 | 3 |
| 3 | 1 | 1 |
| 3 | 1 | 2 |
| 3 | 2 | 3 |
```

SUM()

Syntax:

```
  SUM([DISTINCT | ALL] expression) [OVER (analytic_clause)]
```

Example:

Group in terms of property, and calculate the sum for the current row and the ones before and after the current row and for the column x in the group.

```
  select x, property,
sum(x) over
(
partition by property
order by x
rows between 1 preceding and 1 following
) as 'moving total'
from int_t where property in ('odd','even');
| x | property | moving total |
|----|----------|--------------|
| 2 | even | 6 |
| 4 | even | 12 |
| 6 | even | 18 |
| 8 | even | 24 |
| 10 | even | 18 |
| 1 | odd | 4 |
| 3 | odd | 9 |
| 5 | odd | 15 |
| 7 | odd | 21 |
| 9 | odd | 16 |
```

Notes

Palo supports SQL remarks.

- Single line remark: Statements beginning with -- may be recognized as remarks and ignored. Single-line remarks can be made independently or appear after partial or complete statements of other statements.
- Multi-line remarks: '/*' and' */' text will be recognized as remarks and ignored. A multi-line remark can occupy a single line or multiple lines, or appear in the middle, front, or back of other statements.

Example:

```
 mysql> -- This line is a comment about a query
mysql> select ...
mysql> /*
   /*> This is a multi-line comment about a query.
   /*> */
mysql> select ...
mysql> select * from t /* This is an embedded comment about a query. */ limit 1;
mysql> select * from t -- This is an embedded comment about a multi-line command.
    -> limit 1;
```

SQL-Manual

Palo provides online and offline SQL manuals.

The online SQL manual is viewed by using the Help command after connecting to Palo. For example, view how to create a database.

```
 mysql> help create database;
Name: 'CREATE DATABASE'
Description:
    This statement is used to create a database.
    Syntax:
        CREATE DATABASE [IF NOT EXISTS] db_name;
Examples:
    1. Create a database db_test
        CREATE DATABASE db_test;
```

This document is an offline SQL manual, detailing the syntax of SQL.