PALO Document

2021-12-17

Contents

Contents 2

Function Release Record 5

Product Description 7

- Product Introduction 7
- System Architecture 7

Product Pricing 8

- Billing Instructions 8
- Prepaid 10
- Postpaid 12

Getting Started 14

- Get Started in 5 Minutes 14
- Create Clusters 24

Start to Use 25

- Basic Operations Guide 25
- Advanced Operations Guide 31
- Relation Model and Data Division 32
- Data Model 38
- Index 49

Operating Manual 51

- Data Load 51
  - Load Overview 51
  - Load Local Data 52
  - Load Data in BOS 56
  - Subscribe Kafka Log 61
  - Use JDBC to Synchronize Data 64
  - Synchronize Data Through External Table 67
  - Load JSON Data 70
  - Load Transaction and Atomicity 82
  - Column mapping, converting and filtering 83
  - Strict Mode 89
- Data Update and Delete 92
  - Data Update 92
  - Data Delete 94
  - Mark Deletion 96
  - Sequence-Column 100
- Data Export 106
  - Export Overview 106
  - Full Data Export 106
  - Export Data to External Table 108
  - Export Query Results Set 109

- BI Tool Access 112
  - Sugar 112
  - Navicat 114
  - Tableau 116
  - DBeaver 118
  - YONGHONGBI 119
  - FineBI 122
- Backup and Recovery 126
- Materialized View 130
- Privilege Management 141
- Resource Management 144
- Variable 145
- Time Zone 152
- Experimental Functions 153

Operation Guide 154
- Cluster Scaling 154

SQL Manual 154
- Built-in Functions 154
  - String Function 154
  - Conditional Function 181
  - HLL Function 186
  - Generic Function 191
  - BITMAP Function 192
  - Format Conversion Function 207
  - Time and Date Function 211
  - Type Conversion Function 261
  - Aggregate Function 261
  - Bit Operation Function 272
  - JSON Parsing Function 276
  - Mathematical Function 279
- Syntactical Help 308
  - DDL 308
  - DML 335
  - Information View Statement 376
  - Account Management 408
  - Auxiliary Commands 420
- Alias 425
- Comments 426
- Data Type 426
- Literal Constant 430

SQL Operators 431
SQL-Manual 435
Service Level AgreementSLA 435
Palo Service Level Agreement SLA（V1.0） 435
Open Source Zone 437
Open Source Version 437

Function Release Record

Release time	Feature Overview
2020-12-15	• Query performance is optimized (Join Reorder), query peformance by associating multiple tables has been improved for 100+times, and the memory consumption has been reduced for 5~10 times
2020-11-25	• Query push-down of VALUE column in UNIQUE table, query performance improved 2-100 times • Accelerate the reading of UNIQUE single version and multiple versons, reading performance has 20-40% increase than before
2020-10-27	• Palo UI was published, supporting logging in Web UI from console, quick connection to clusters and query
2020-07-23	• Support INTERSECT and EXCEPT operators • Support setting SQL grammer by group, can further reduce the complexity of SQL that programmed for data analysts.
2020-05-19	• Support automatic creation of partition through planned tasks, reduce the maintenance cost of partition operation for users
2020-04-26	• Support bitmap index, improve the query performance of Palo • Support Spark on Palo, support query of data stored in Palo through Spark • Support materialized view, and analysis of original detail data at any dimension
2019-12-25	• Support import Parquet file through broker • Support time zone, users can specify time zone when querying or loading
2019-10-29	• Reconstruct storage, support storage of files of more formats • Support joint operation of bitmap types and bitmaps
2019-07-18	• Support the joint query of tables in Palo on ES, Palo and ES, and more complex full text search and filter
2019-06-02	• Support regular import function, which automatically imports data from specified data source • Support UDF and UDAF
2019-02-18	• Support streaming import, import throughout at single node can reach 100MB/s • Introduce RocksDB to store the meta information of sharding, greatly reduce the random IO operations
2018-09-28	• Support local Join operation of data in multiple tables, reduce the delay of Join query
2018-06-18	• The compatibility with MySQL protocol was enhanced

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.

Privatization deployment

Palo not only provides fully hosted data warehouse services on Baidu AI cloud, but also supports deployment and support services on the self-built data centers or private clouds of enterprises. For details, please visit Privatization Deployment Consultation.

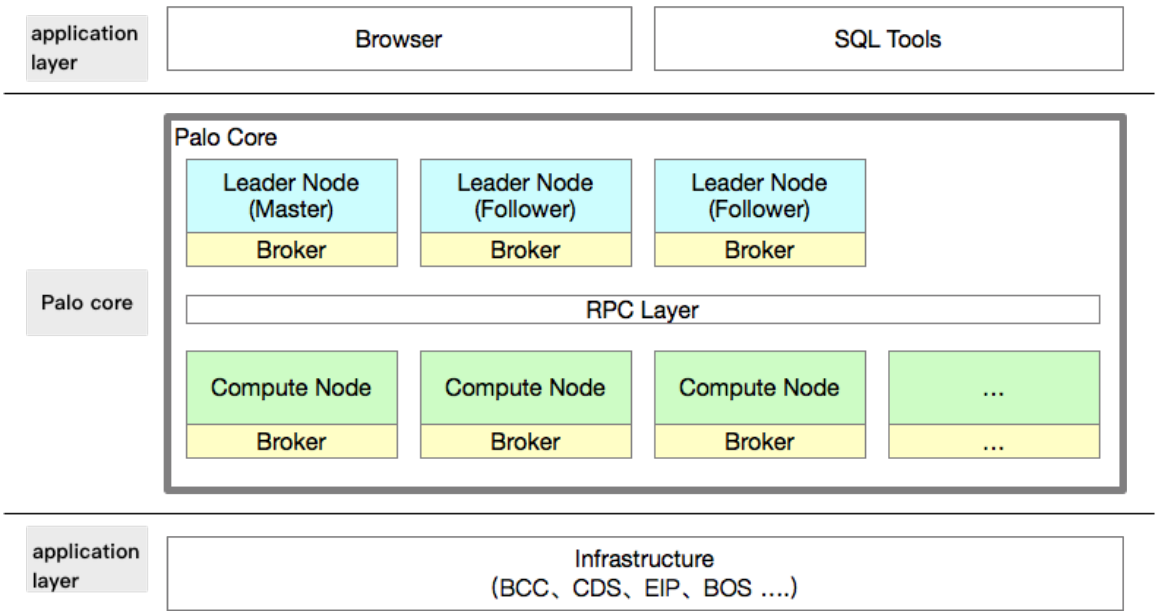
Open source version

Apache Doris(Incubating) is an incubating project opened by Baidu Palo team and contributed to Apache Foundation. Users may view and download it in Open Source Special Zone for use.

System Architecture

Palo has a very simple and elegant system architecture. Its core components only include two kinds of components, one with state, one with no state which is optional. When using the Palo service hosted by Baidu AI cloud, an extra front-end console is required for some cluster management operations.

Architecture diagram



As shown in the above figure, the middle part is the core component of Palo, mainly including the following modules:

1. Leader Node

   Leader Node is also called Frontend（FE）. In other documents, both of the names refer to the same componnet. The Leader Node is mainly used for accessing the connection requested by users, storage and management of metadata, parsing of query statements and generation of query plans, management of the cluster nodes status, as well as the tasks scheduling submitted by various users and within the system.

   The Leader Node has built-in MySQL protocol layer and Http Server. Users can choose different protocols to connect to the

Leader Node depending on specific operation. At the same time, the Leader Node also has a built-in UI interface. Users can click the **Palo UI** button at the upper right corner of the front console page to enter.

Generally, one Palo cluster has 1 or 3 Leader Nodes. When the number is 3, Leader Node will automatically form a node group to secure the high availability of components.

2. Compute Node

   Compute Node is also called Backend (BE). In other documents, both of the names refer to the same component. Compute Node is mainly used for the execution of query plan and the storage and management of data. Compute Node can be composed of one or more nodes, and the overall computing power and storage capacity of Palo cluster are in direct proportion to the number of of Compute Node.

   Leader Node interacts with Compute Node through RPC.

3. Broker

   Broker is a stateless process that helps Palo to access the external data source by means of Unix-like file system interface, for example, the data on BOS or HDFS and apply them in data import or export operations.

   In the Palo cluster hosted by Baidu AI cloud, this component will be installed by default. You can view it by executing the `SHOW BROKER` command after connecting to MySQL protocol by using the MySQL protocol. When in privatization deployment scenario, this component can be an optional component.

Product Pricing

Billing Instructions

Charging items

Baidu data repository Palo only charges for the resources of hosting Palo cluster, and no additional fees are charged for other cloud supporting functions such as monitoring. Charging items of Palo are as follows:

Charging items	Instructions
Cluster nodes	- provided in the form of node specifications, including CPU, memory and disk storage. - Supporting advance payment and post-payment.

The rest of public cloud products by means of matched use and in need of additional payment in posse include but are not limited to:

Products	Instructions
EIP	Elastic public IP. When accessing Palo cluster from public network, the user needs to purchase EIP and bind it.
BOS	Baidu object storage, it will be charged according to the flow and storage space

Refer to Here for **details of privatization deployment charges.

Charging mode

Palo supports advance payment (yearly or monthly charging) and post-payment (charging by quantity). The detailed charging rules are as follows:

Charging methods	Instructions	Points for attention
Advance payment - yearly or monthly charging	- i. e. yearly or monthly charging, making required payment when creating the instance. - Suitable for the long-term demand of business stability with lower cost than post-payment, and the longer the purchase time is, the more the discount will be.	- The advance payment instance does not support unsubscribing, if you need to release, please submit the work order for processing.
Post-payment – charging by quantity	- Charging by minute, deducting the charge and generate the bill in Beijing time hourly according to the instance. - The billing time is within 1 hour after the end of the current pricing cycle. - Suitable for the scenario where the business has great changes in an instant, and the resources will be released immediately after use to reduce the cost.	- Post-payment instances can be released at any time. - Before purchase, it is necessary to ensure that there is no arrears in the account, and the total amount of account balance and available vouchers is greater than or equal to 100 yuan.

Payment methods

Make choice between the following payment methods when purchasing Palo:

- Baidu Intelligent Cloud account balance, refer to Management console - Financial overview for current balance.
- Online payment.
- Voucher payment, please make sure that the voucher status is in use and the order type is general. Log on to Management console - Financial center - Voucher for list of currently owned vouchers.

Charging changes

Baidu Intelligent Cloud Palo supports charging mode switching service to meet the needs of users, and the two charging methods of on-demand charging (post-payment) and monthly or yearly charging (advance payment) can be flexibly switched.

Changing advance payment to post payment

Description of charging changes

1. Advance payment orders need to be switched 24 hours before the due date, and the changed charging method will take effect after the expiration of the current order, while the due time is subject to the financial order.
2. Charging changes are not allowed within 24 hours prior to due time.
3. When the advance payment is switched to post-payment, the charging method will take effect after the service expires.
4. This function only supports users who have not opened automatic renewal.

> Note: when advance payment is changed to post-payment, the account balance must be greater than 100 yuan, if it is insufficient, the order will fail and void, and the way of charging switch will also fail.

Operation steps

1. Log on to the console, select "Product Services - > Data Repository Palo" to enter the cluster management page, select Palo cluster requiring to change the charging method, and select the item charging change in "More operations".

2. Enter the "Charging change" page to confirm the node and financial information after the advance payment is changed to post-payment. If there is no problem, click the button "Next step".
3. Confirm the order information, if there is no problem, click the button "Pay".
4. When the payment is completed, the change of advance payment to post- payment is completed.
5. For cluster instances that have completed charging changes, a sign will remind users which clusters have set charging change.
6. Before the charging change takes effect, the user can cancel the previous charging change at any time. Check the cluster column and select the item "Cancel pricing change" in "More operations".
7. If you cancel the charging change, a confirmation box will pop up, click the button "Confirm" to cancel the change.
8. After confirming the cancellation of the change, the charging change can be made when necessary, but note that the charging change should be operated 24 hours before the cluster expires, and the cluster that expires within 24 hours does not allow the charging change.

Changing post-payment to advance payment

Description of charging changes

1. Changing post-payment to advance payment can be made at any time;
2. Advance payment will take effect immediately after changing.

Operation steps

1. Log on to the console, select "Product Services - > Data Repository Palo" to enter the cluster management page, select Palo cluster requiring to change the charging method to post-payment, and select the item charging change in "More operations".
2. On the charging change page, select advance payment duration, confirm and click the button "Next step".
3. The advance payment process of charging change is consistent with the process of normal cluster purchase, click the button "pay" to make payment and click the button "Confirm payment".
4. After the payment, the charging change is completed.
5. Go back to the cluster management list, the previous post-payment cluster has been changed to advance payment cluster can be seen. Since the change from post-payment to advance payment takes effect immediately, the cluster has no special prompt.

Prepaid

This document introduces the pricing, charging rules and expiration reminder for advance payment of Baidu data repository Palo.

During advance payment, the user can prepay the usage fee of Palo monthly and Palo will calculate the monthly package price according to the configuration and number of instance models chosen by the user. Generally, the advance payment price for the same duration is far lower than on-demand payment price.

> Note: before purchasing, it is necessary to ensure that the account is free of arrears.

Advance payment Charging formula

Cost = unit price × number of nodes × usage time

> Advance payment discount: 83.0% off for one year, 70% off for two years and 50% off for three years.

Node package specifications

- **Balanced 1 type**

Node configuration	CPU (Core)	Memory (GB)	SSD Cloud disk (GB)	Unit price (Yuan/month/node)
Leader Node	4	16	100	700
Compute Node	4	16	200	800

- **Balanced 2 type**

Node configuration	CPU (Core)	Memory (GB)	SSD Cloud disk (GB)	Unit price (Yuan/month/node)
Leader Node	8	32	200	1400
Compute Node	8	32	500	1700

- **Balanced 3 type**

Node configuration	CPU (Core)	Memory (GB)	SSD Cloud disk (GB)	Unit price (Yuan/month/node)
Leader Node	16	32	200	1688
Compute Node	16	64	200	2428

- **Storage 1 type**

Node configuration	CPU (Core)	Memory (GB)	SSD Cloud disk (GB)	Unit price (Yuan/month/node)
Leader Node	8	32	200	1319
Compute Node	8	32	1024	2184

- **Storage 2 type**

Node configuration	CPU (Core)	Memory (GB)	SSD Cloud disk (GB)	Unit price (Yuan/month/node)
Leader Node	16	32	200	1688
Compute Node	32	64	4096	7258

- **Storage 3 type**

Node configuration	CPU (Core)	Memory (GB)	High performance cloud disk (GB)	Unit price (Yuan/month/node)
Leader Node	16	32	200	1688
Compute Node	16	64	4096	3723

- **Storage 4 type**

Node configuration	CPU (Core)	Memory (GB)	High performance cloud disk (GB)	Unit price (Yuan/month/node)
Leader Node	16	32	200	1688
Compute Node	16	64	2048	2970

- **Calculation 1 type**

Node configuration	CPU (Core)	Memory (GB)	SSD Cloud disk (GB)	Unit price (Yuan/month/node)
Leader Node	16	32	200	1688
Compute Node	32	64	500	3482

> Note: the configurations that can be selected for different types and regions of users may be different, and the configurations that can be purchased are displayed in Console.

Expiration reminder

- The system will send a reminder notice about the expiration 7 days before the expiration of Palo service.
- The service will stop immediately after expiration, and the system will send a notice of service suspension for arrears. If the service has not been renewed after 7 days, the system will automatically delete the service and release resources.

Postpaid

Post-payment means paying by usage, in this mode, you can pay for the usage of Palo according to the usage duration and can stop paying at any time to release the cluster.

> It is necessary, before purchasing the cluster, to ensure that there is no arrears in the account, and that the sum of the account balance and the available vouchers is greater than or equal to 100 yuan.

Post-payment charging formula

Cost = unit price × number of nodes × usage time

Node package specifications

- **Balanced 1 type**

Node configuration	CPU (Core)	Memory (GB)	SSD cloud disk (GB)	Unit price (Yuan/min/node)
Leader Node	4	16	100	0.0345
Compute Node	4	16	200	0.0370

- **Balanced 2 type**

Node configuration	CPU (Core)	Memory (GB)	SSD Cloud disk (GB)	Unit price (Yuan/min/node)
Leader Node	8	32	200	0.0691
Compute Node	8	32	500	0.0764

- **Balanced 3 type**

Node configuration	CPU (Core)	Memory (GB)	SSD Cloud disk (GB)	Unit price (Yuan/min/node)
Leader Node	16	32	200	0.0901
Compute Node	16	64	200	0.1332

- **Storage 1 type**

Node configuration	CPU (Core)	Memory (GB)	SSD Cloud disk (GB)	Unit price (Yuan/min/node)
Leader Node	8	32	200	0.0691
Compute Node	8	32	1024	0.0893

- **Storage 2 type**

Node configuration	CPU (Core)	Memory (GB)	SSD Cloud disk (GB)	Unit price (Yuan/min/node)
Leader Node	16	32	200	0.0901
Compute Node	32	64	4096	0.2707

- Storage 3 type

Node configuration	CPU (Core)	Memory (GB)	High performance cloud disk (GB)	Unit price (Yuan/min/node)
Leader Node	16	32	200	0.0901
Compute Node	16	64	4096	0.1635

- Storage 4 type

Node configuration	CPU (Core)	Memory (GB)	High performance cloud disk (GB)	Unit price (Yuan/min/node)
Leader Node	16	32	200	0.0901
Compute Node	16	64	2048	0.1460

- Calculation 1 type

Node configuration	CPU (Core)	Memory (GB)	SSD Cloud disk (GB)	Unit price (Yuanmin/node)
Leader Node	16	32	200	0.0901
Compute Node	32	64	500	0.1826

> Note: the configurations that can be selected for different types and regions of users may be different, and the configurations that can be purchased are displayed in Console.

Charging rules

According to the configuration and number of instance models you choose, Baidu data repository Palo charges according to the usage time (minute level). Specific pricing rules are as follows:

- Charging by minute, and less than one minute is counted as one minute.
- Charging starts when the order is submitted and the cluster instance is in the `running` state, .
- Charging stops when the cluster is deleted and all resources are released immediately.
- Charging still continues when the cluster is stopped because the resources have not been released; if the charging needs to be stopped, delete the cluster and release the resources.

Insufficient balance and arrears

Reminder of insufficient balance :

- According to the amount of your bill in the last three days, the system judges whether your account balance (including available vouchers) is enough to pay for the next three days. If not, the system will send a renewal reminder.
- According to the amount of your bill in the latest day, the system judges whether your account balance (including available vouchers) is enough to pay for the next day . If not, the system will send a renewal reminder.

Treatment of arrears :

- When your account balance is 0 yuan and you cannot pay the Palo service bill, you are in arrears. And the system will send a notice of arrears.
- The service will stop immediately after arrears, and the system will send a notice of arrears. In order not to affect your service, it is recommended that you make sure that there is enough money in your account is enough to pay for normal service when using Palo service.
- After 7 days of arrears, the system will automatically delete the service and release resources if the account has not been recharged.

Getting Started

Get Started in 5 Minutes

In this tutorial document, we will introduce how to use Palo UI to have a quick experience and how to use Palo to query.

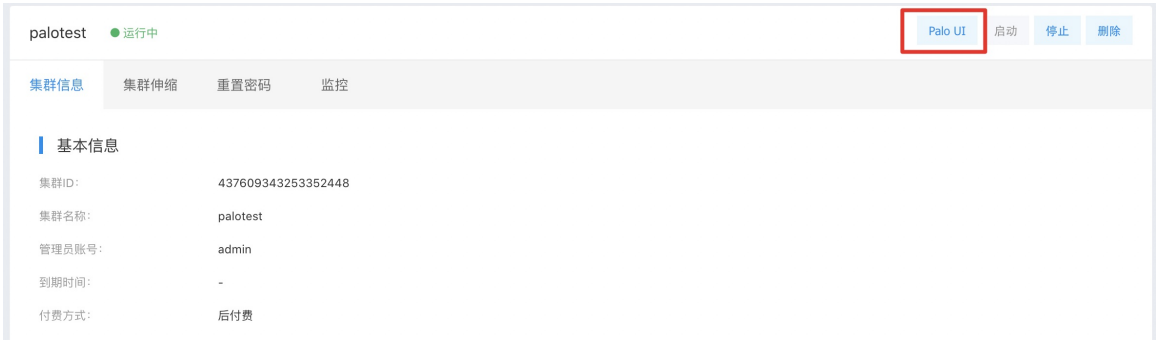
> Before reading this tutorial ducment, please refer to Create Cluster document to create a PALO cluster.

Palo UI is a Web UI environment that Palo provides to quickly execute query requests provided by Palo, which can quickly execute query requests and perform some management operations.

> The demo data and query examples used in this document are all from Star Schema Benchmark, user can clickDownload to get sample data and SQL statements.

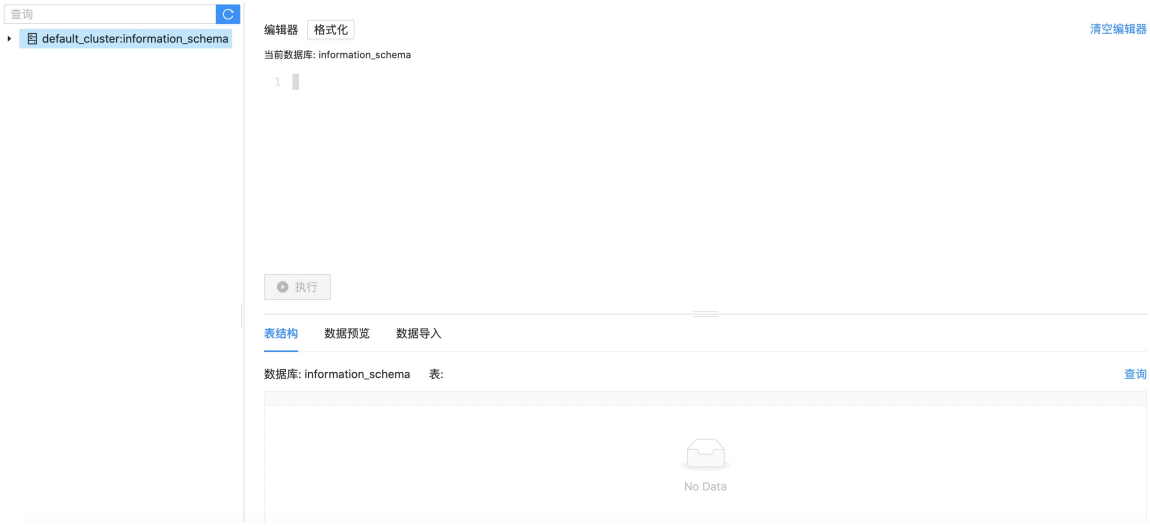
Enter into Palo UI

In the details page of Palo cluster, a quick entry for Palo UI is provided.



Click the Palo UI button to enter the Web UI environment. Enter the user name and password on the login page, the user name is "admin" and the password is the password filled in when the cluster is created.

Click login to enter the main page of Palo UI, the default page is Palo query page (Playground).



Palo quick query page is mainly divided into three areas, and the left side is the table management area, including system library tables and tables created by users themselves. The upper right area is the SQL execution area, and the lower right area is the table preview, data import and execution result area. Next, we show the main steps from database building, table building, data importing, query, etc. on this page to help users who use Palo for the first time experience a complete use process.

Database building and table building

In the editor area, enter SQL statements to create a library of `example_db`. After clicking the Execute, an execution result will be displayed at the bottom. After successful execution, refresh the left table area, the newly created `example_db` library will be displayed in the table management area.

```
CREATE DATABASE example_db;
```

The demo data contain a total of 5 sales-related orders, dates, customer information and other data samples, so we need to create 5 corresponding tables.

Firstly, in the `example_db` library, create a table named `lineorder`. Palo uses the key word `DISTRIBUTED` keyword to set the

bucketing column used to divide the data horizontally. Generally, we choose a column that can help data to be evenly divided as the bucketing column. Here we use `lo_orderkey` as the bucketing column. Here we also set the copy number as 1, as the default copy number in Palo is three, if we buy one Compute Node in the cluster, we need to set the copy number as 1.

This table mainly defines the order number, order time, profit, taxes and other order main information.

```
CREATE TABLE lineorder (
    lo_orderkey BIGINT,
    lo_linenumber BIGINT,
    lo_custkey INT,
    lo_partkey INT,
    lo_suppkey INT,
    lo_orderdate INT,
    lo_orderpriotity VARCHAR(16),
    lo_shippriotity INT,
    lo_quantity BIGINT,
    lo_extendedprice BIGINT,
    lo_ordtotalprice BIGINT,
    lo_discount BIGINT,
    lo_revenue BIGINT,
    lo_supplycost BIGINT,
    lo_tax BIGINT,
    lo_commitdate BIGINT,
    lo_shipmode VARCHAR(11)
)
DISTRIBUTED BY HASH(lo_orderkey)
PROPERTIES ("replication_num"="1");
```

Then we create a `date` table. Using `d_datekey` as bucketing column, and set the copy number as 1. This table defines more detailed order date information.

```
CREATE TABLE date (
  d_datekey INT,
  d_date VARCHAR(20),
  d_dayofweek VARCHAR(10),
  d_month VARCHAR(11),
  d_year INT,
  d_yearmonthnum INT,
  d_yearmonth VARCHAR(9),
  d_daynuminweek INT,
  d_daynuminmonth INT,
  d_daynuminyear INT,
  d_monthnuminyear INT,
  d_weeknuminyear INT,
  d_sellingseason VARCHAR(14),
  d_lastdayinweekfl INT,
  d_lastdayinmonthfl INT,
  d_holidayfl INT,
  d_weekdayfl INT
) DISTRIBUTED BY hash(d_datekey) PROPERTIES (
 "storage_type"="column",
 "replication_num"="1");
```

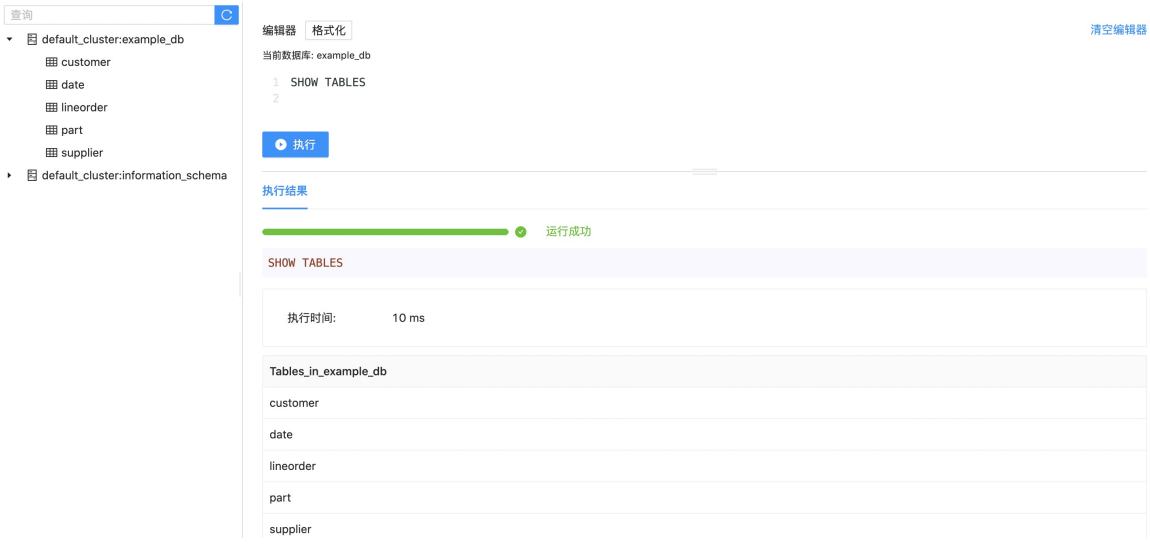
Next, we create the remaining three tables, `customer`,`part`,`supplier` which record the detailed information of customers, goods and suppliers respectively.

```
 CREATE TABLE customer (
    c_custkey INT,
    c_name VARCHAR(26),
    c_address VARCHAR(41),
    c_city VARCHAR(11),
    c_nation VARCHAR(16),
    c_region VARCHAR(13),
    c_phone VARCHAR(16),
    c_mktsegment VARCHAR(11) )
DISTRIBUTED BY hash(c_custkey)
PROPERTIES (
  "storage_type"="column",
  "replication_num"="1");

CREATE TABLE part (
    p_partkey INT,
    p_name VARCHAR(23),
    p_mfgr VARCHAR(7),
    p_category VARCHAR(8),
    p_brand VARCHAR(10),
    p_color VARCHAR(12),
    p_type VARCHAR(26),
    p_size INT,
    p_container VARCHAR(11) )
DISTRIBUTED BY hash(p_partkey)
PROPERTIES (
  "storage_type"="column",
  "replication_num"="1");

CREATE TABLE supplier (
    s_suppkey INT,
    s_name VARCHAR(26),
    s_address VARCHAR(26),
    s_city VARCHAR(11),
    s_nation VARCHAR(16),
    s_region VARCHAR(13),
    s_phone VARCHAR(16) )
DISTRIBUTED BY hash(s_suppkey)
PROPERTIES (
  "storage_type"="column",
  "replication_num"="1");
```

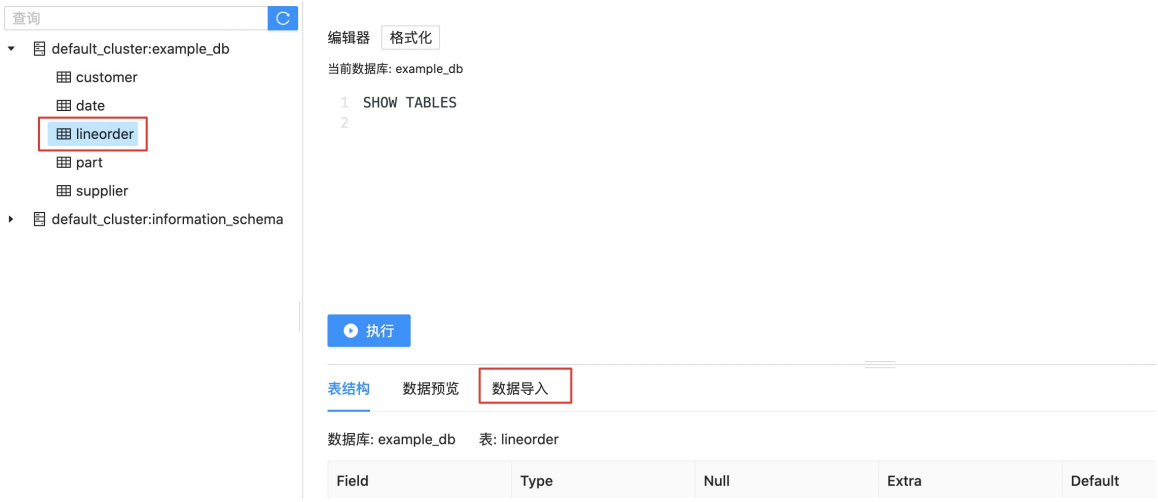
After the table is built, you can view the information of the table in example_db:



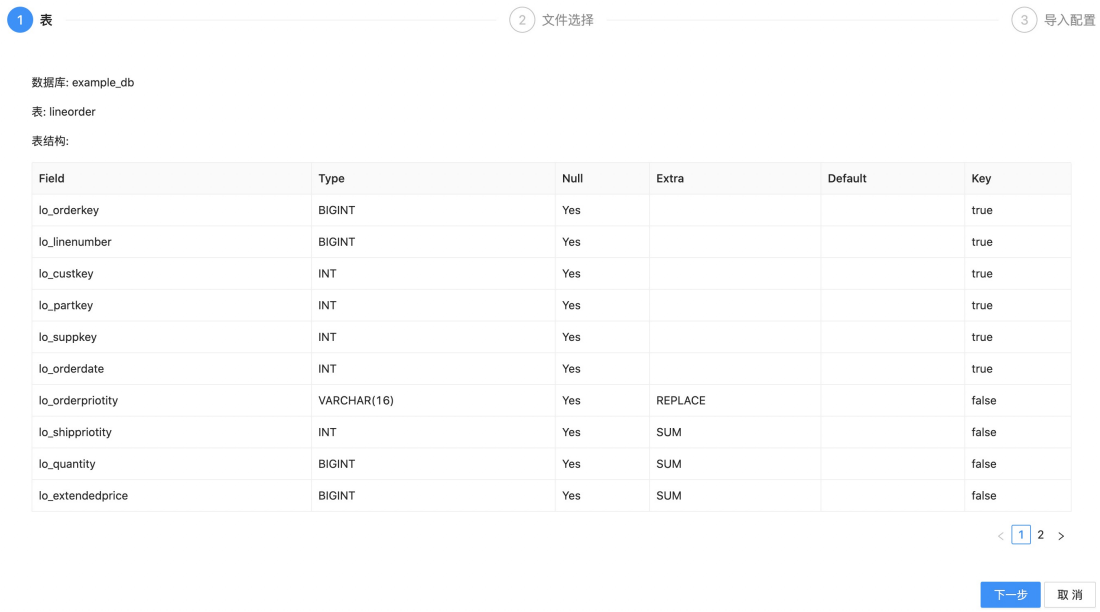
Import data

Palo supports multiple data import methods. For details, refer to data import document. Here, we use Web to import data conveniently as an example.

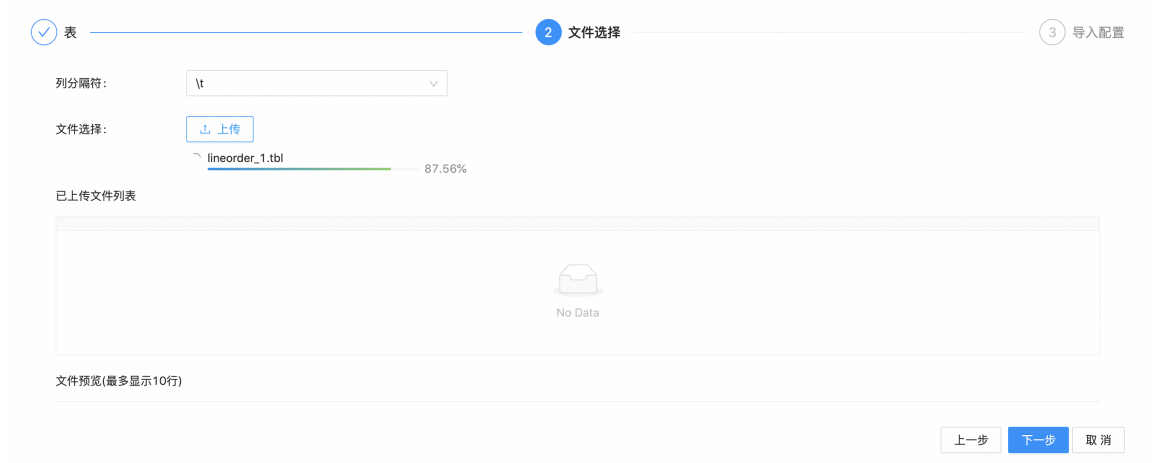
First, click to Select the table to import data



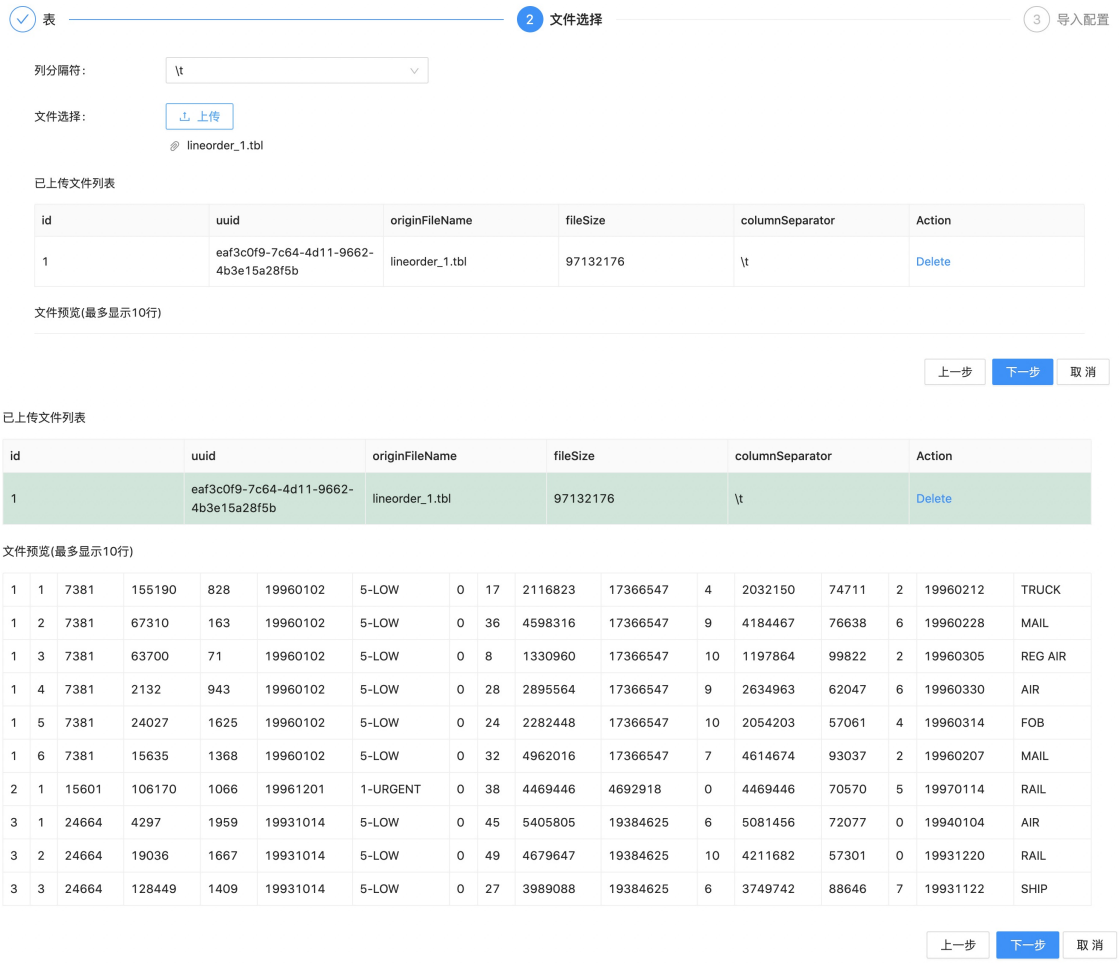
Then click Data Import to enter the data import page



Click "Next". Select the column separator of the imported file, here we use `\t` . Then select the data file to import

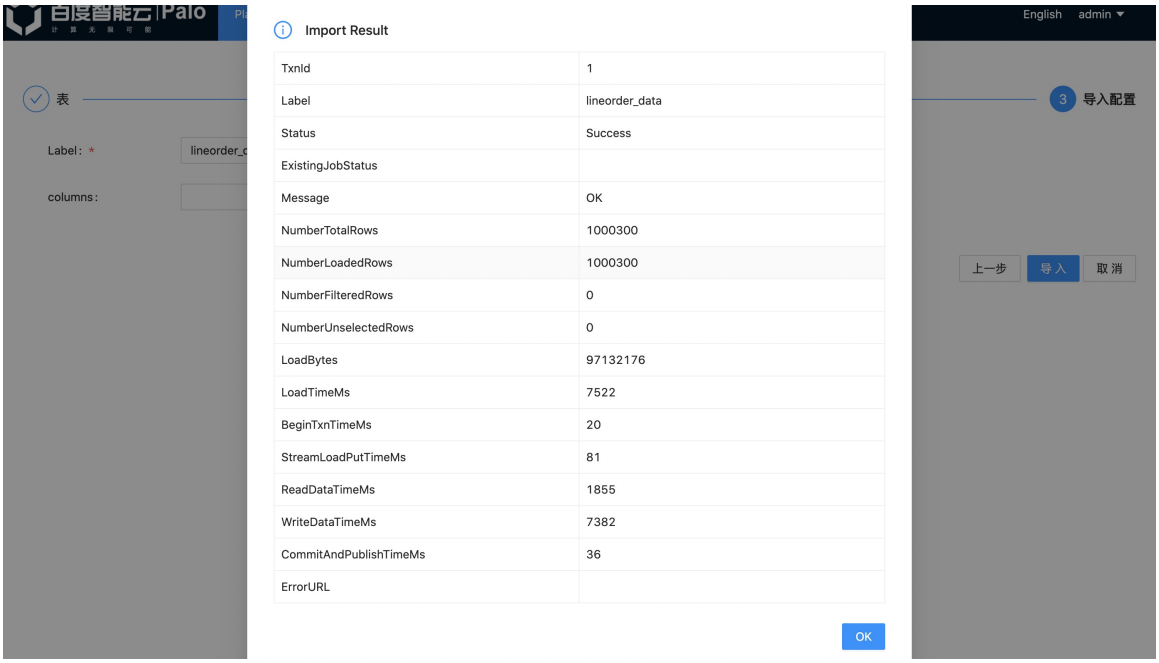


Wait for the file loading, then click to select the uploaded file below. Then the preview of the file data separated by specified separator (first 10 lines) will be displayed.



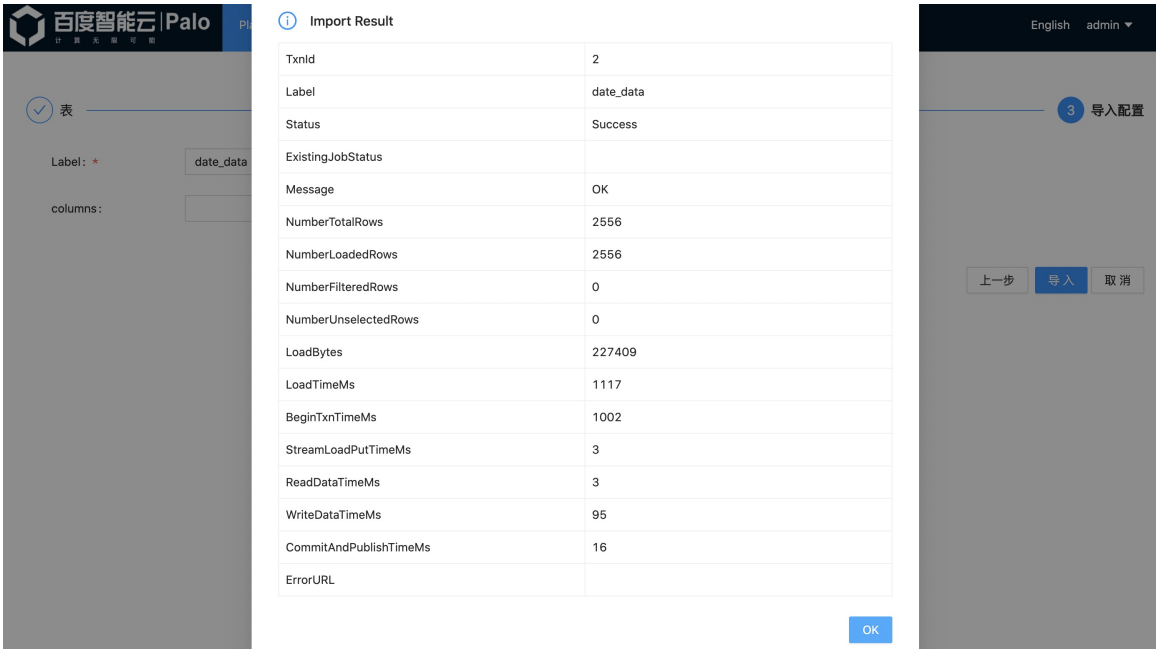
Click "Next" to enter the data import configuration page. We use "lineorder_data" as label, and then click "import" button.

Wait for a moment, then the result of data import will be displayed. In which, the `Status` will change to `Success`, indicating that the import is successful. After clicking "OK", the data import is completed.



As the size of data imported by Web UI is limited, we divide the complete lineorder data into six copies, here we only import one copy of data as a demonstration and put complete data samples in the demo files, users can add and import all the data according to the test requirements.

The data corresponding to tables`date`, `customer`, `part`, `supplier` are imported in the same way.



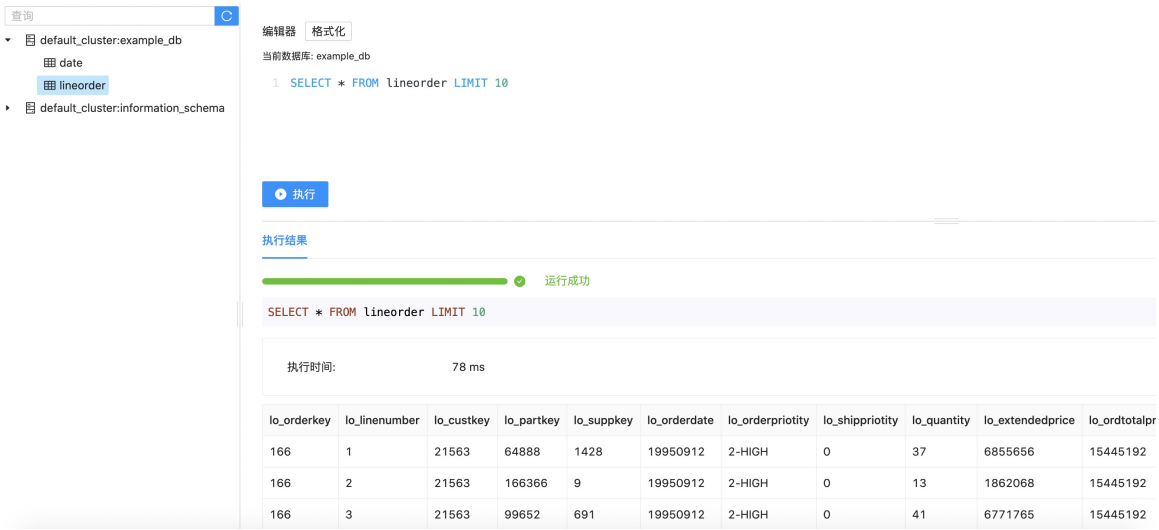
Data Query

Simple search

After the data import is completed, we can execute some query statements to check the status of the data.

Part of data in the table can be previewed.

```
SELECT * FROM lineorder limit 10
```



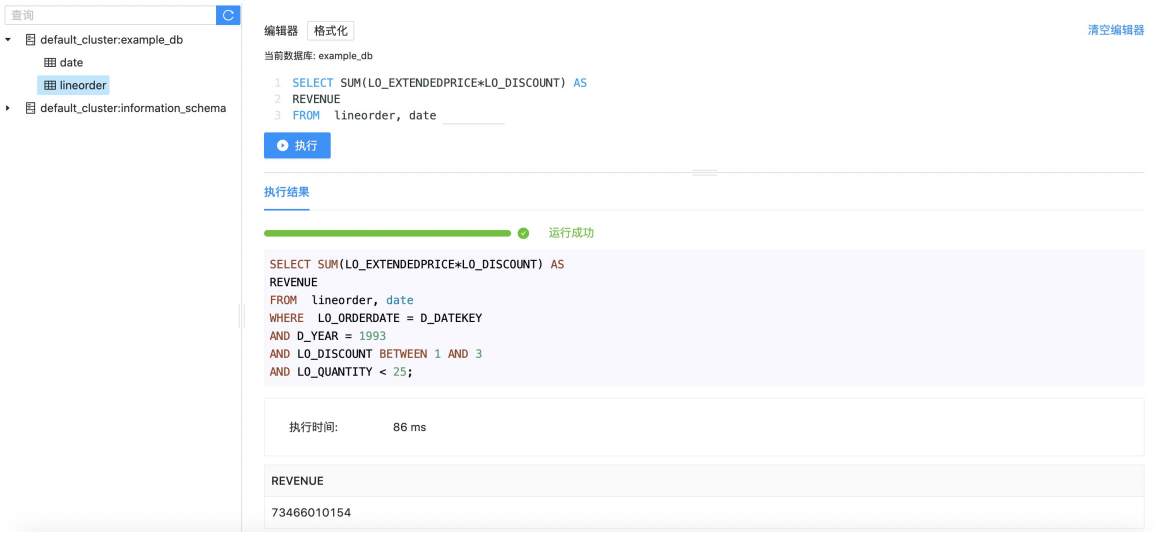
Or count the number of table query recorded.

```
SELECT COUNT(*) FROM lineorder
```

Then execute the query operation according to our analysis requirements and obtain the query results.

```
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;
```

Through Web UI, we execute SQL on the page and obtain the query results quickly.



Complex query of multiple tables can also be executed.

```
 SELECT C_CITY, S_CITY, D_YEAR, SUM(LO_REVENUE)
AS  REVENUE
FROM customer, lineorder, supplier, date
WHERE  LO_CUSTKEY = C_CUSTKEY
AND LO_SUPPKEY = S_SUPPKEY
AND  LO_ORDERDATE = D_DATEKEY
AND C_NATION = 'UNITED STATES'
AND S_NATION = 'UNITED STATES'
AND D_YEAR >= 1992 AND D_YEAR <= 1997
GROUP BY C_CITY, S_CITY, D_YEAR
ORDER BY D_YEAR ASC,  REVENUE DESC;
```

Execution time and results of page query:

运行成功

```
SELECT C_CITY, S_CITY, D_YEAR, SUM(LO_REVENUE)
AS  REVENUE
FROM customer, lineorder, supplier, date
WHERE  LO_CUSTKEY = C_CUSTKEY
AND LO_SUPPKEY = S_SUPPKEY
AND  LO_ORDERDATE = D_DATEKEY
AND C_NATION = 'UNITED STATES'
AND S_NATION = 'UNITED STATES'
AND D_YEAR >= 1992 AND D_YEAR <= 1997
GROUP BY C_CITY, S_CITY, D_YEAR
ORDER BY D_YEAR ASC,  REVENUE DESC;
```

执行时间: 58 ms

C_CITY	S_CITY	D_YEAR	REVENUE
UNITED ST7	UNITED ST0	1992	28216906
UNITED ST9	UNITED ST1	1992	22978097
UNITED ST1	UNITED ST0	1992	21978307
UNITED ST5	UNITED ST1	1992	21929161
UNITED ST0	UNITED ST1	1992	21291177

> The execution time is the actual time consumed by SQL at server end. As there are many layers of agency on UI interface, the query delay sensed by users will be slower than the actual execution time of SQL.

So far, we have completed a complete proces of database and tables building, data import and query. For more operations, please refer to **Get Started**.

Create Clusters

If you don't have an account of Baidu Open Cloud, please first register a Baidu account Here, then use your account to log in Baidu Open Cloud and enter into Palo main page:

After entering, click `Buy Now`:

If it is the first time you create a Palo cluster, the following interface will pop up when you enter the Palo cluster list page:

Please read the tips on the page, and click `Open` .

Then you can enter into the cluster list page.

Click `Create Cluster` to enter the cluster purchase page:

Fill in the cluster name and set the cluster password. The node network and available zone can be customized according to your other services on Baidu Cloud and use the default without configuration if there is no requirement. For more details about VPC, click Here.

After filling in the basic information, configure the nodes. The nodes of Palo are divided into Compute Node and Leader Node, in which Leader Node is the main node, mainly used for metadata management, query analysis and query plan generation, as well as cluster task scheduling job management, and can be set to 1 or 3. In production environment, it is recommended to set 3 leader nodes. Compute Node is mainly used for data storage and query plan execution, which is generally configured according to data volume and query requirements, and can be expanded later. At present, the upper limit of Compute Node is 100, which can support PB-level data volume query.

After completing the configuration, click `Create Cluster` to enter into the unpaid page :

Click `Go to Pay` to enter into the payment page. After the payment is completed, the opening success page will be displayed:

The creation of Palo cluster can be completed in about 1-5 minutes. You can click `Management Console` to enter into the console and view the cluster being created or already created.

Start to Use

Basic Operations Guide

In the quick start tutorial, we completed some basic operations of Palo through UI interface of Palo. In the actual production environment, we usually need to connect to Palo and carry out various operations with the programs.

Palo uses MySQL protocol for communication, so users can use standard MySQL client, or MySQL library, JDBC, ODBC and other languages to connect Palo. This paper takes MySQL client as an example to show the basic usage of Palo through a complete process.

> MySQL client with the version after 5.1 is recommended when choosing MySQL client, because the user name with a length of more than 16 characters cannot be supported by the version before 5.1.
>
> Download the MySQL client of Linux version here: mysql-5.7.22-linux-glibc2.12-x86_64. (After decompression, there is a `mysql` binary program in `bin/` directory.)

Creating users and databases

The password set by the user when creating the Palo cluster is the password of the Palo admin user. By default, Palo cluster initially contains an admin user, and the user can connect with Palo for the first time through the admin user.

```
mysql -hPALO_HOST -PPALO_PORT -uadmin -pyour_password
```

For MySQL clients of version 8.0 or above, please add the following parameters:

```
mysql --default-auth=mysql_native_password -hPALO_HOST -PPALO_PORT -uadmin -pyour_password
```

The host and port here are the **MySQL protocol connection targets** given on the Palo console page. If the user is bound with EIP, replace it with EIP.

> Note: for Baidu intranet special users, please contact student on duty of Palo obtain the IP address of the connection target.

The admin user has all the operation privileges of the cluster. Only administrators are recommended to use. Administrators can use the admin user to create ordinary users and grant corresponding privileges.

Create an ordinary user through the following command:

```
CREATE USER 'jack' IDENTIFIED BY 'jack_passwd';
```

The newly-created ordinary user does not have any privileges by default. We can then create a database and authorize the user Jack.

```
CREATE DATABASE example_db;

GRANT ALL ON example_db to "jack";
```

After that, we can use user jack to log in and view the database.

```
mysql -hPALO_HOST -PPALO_PORT -ujack -pjack_password
```

```
MySQL> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| example_db         |
| information_schema |
+--------------------+
2 rows in set (0.00 sec)
```

Create a table

Switch the database first:

```
USE example_db;
```

Next create a table:

```
CREATE TABLE lineorder (
    lo_orderkey BIGINT,
    lo_linenumber BIGINT,
    lo_custkey INT,
    lo_partkey INT,
    lo_suppkey INT,
    lo_orderdate INT,
    lo_orderpriotity VARCHAR(16),
    lo_shippriotity INT,
    lo_quantity BIGINT,
    lo_extendedprice BIGINT,
    lo_ordtotalprice BIGINT,
    lo_discount BIGINT,
    lo_revenue BIGINT,
    lo_supplycost BIGINT,
    lo_tax BIGINT,
    lo_commitdate BIGINT,
    lo_shipmode VARCHAR(11)
)
DISTRIBUTED BY HASH(lo_orderkey)
PROPERTIES ("replication_num"="1");
```

Here we create a `lineorder` table in Star Schema Benchmark with `lo_orderkey` being bucket column，and set the number of copies to 1.

> By default, the number of copies is 3, when the number of BE nodes in the cluster is less than 3, the number of copies should not be greater than the number of BE nodes.

After the table is created, we can view the information of table example_db:

```
MySQL> SHOW TABLES;
+---------------------+
| Tables_in_example_db |
+---------------------+
| lineorder           |
+---------------------+
2 rows in set (0.01 sec)

mysql> DESC lineorder;
+-----------------+-------------+------+-------+---------+-------+
| Field           | Type        | Null | Key   | Default | Extra |
+-----------------+-------------+------+-------+---------+-------+
| lo_orderkey     | BIGINT      | Yes  | true  | NULL    |       |
| lo_linenumber   | BIGINT      | Yes  | true  | NULL    |       |
| lo_custkey      | INT         | Yes  | true  | NULL    |       |
| lo_partkey      | INT         | Yes  | false | NULL    | NONE  |
| lo_suppkey      | INT         | Yes  | false | NULL    | NONE  |
| lo_orderdate    | INT         | Yes  | false | NULL    | NONE  |
| lo_orderpriotity | VARCHAR(16) | Yes  | false | NULL    | NONE  |
| lo_shippriotity | INT         | Yes  | false | NULL    | NONE  |
| lo_quantity     | BIGINT      | Yes  | false | NULL    | NONE  |
| lo_extendedprice | BIGINT     | Yes  | false | NULL    | NONE  |
| lo_ordtotalprice | BIGINT     | Yes  | false | NULL    | NONE  |
| lo_discount     | BIGINT      | Yes  | false | NULL    | NONE  |
| lo_revenue      | BIGINT      | Yes  | false | NULL    | NONE  |
| lo_supplycost   | BIGINT      | Yes  | false | NULL    | NONE  |
| lo_tax          | BIGINT      | Yes  | false | NULL    | NONE  |
| lo_commitdate   | BIGINT      | Yes  | false | NULL    | NONE  |
| lo_shipmode     | VARCHAR(11) | Yes  | false | NULL    | NONE  |
+-----------------+-------------+------+-------+---------+-------+
17 rows in set (0.02 sec)
```

Loading data

Palo supports a variety of data loading methods. Please refer to the data loading document for details. Here we take Broker loading as an example.

Broker loading reads data from external storage for loading through built-in Broker process in the cluster. Please refer to the introduction of Broker in the operation manual for more help.

In order to use Broker loading, we need to store the loaded data files on Baidu object storage BOS in advance. Here we prepare data of `lineorder` table (about 100MB) and we can download and upload them to our own BOS to start loading.

lineorder data examples

Refer to Data loaded to BOS for detailed documents on how to upload to and load data from BOS.

Suppose that the BOS path where the user stores the loading file is:`bos://example_bucket/lineorder_1.tbl`

Then we can load data through the following command:

```
LOAD LABEL example_db.my_first_load
(
    DATA INFILE("bos://example_bucket/lineorder_1.tbl")
    INTO TABLE lineorder
)
WITH BROKER 'bos'
(
    "bos_endpoint" = "http://bj.bcebos.com",
    "bos_accesskey" = "xxxxxxxxxxxxxxxxxxxxxxxxxxxx",
    "bos_secret_accesskey"="yyyyyyyyyyyyyyyyyyyy"
);
```

1. The Broker name of the public cloud Palo cluster is BOS. Refer to `SHOW BROKER;` to view.
2. The `bos_endpoint` in tis example is `http://bj.bcebos.com`. Users need to specify different endpoints according to different regions.
3. accesskey and secret_accesskey can be viewed in public cloud security certification center.

Broker loading is an asynchronous command. Successful execution of the above command only means successful submission of the task.

Whether the loading is successful or not depends on the loading label, view through `SHOW LOAD`command. The label in this example is `my_first_load` :

```
SHOW LOAD WHERE LABLE = "my_first_load";
```

In the returned results, if the `State` field is FINISHED, the loading is successful. Then we can query the data.

Data query

Palo supports SQL syntax in most SQL 92 and SQL 99 standards, as well as some SQL 2003 standards, which, basically, covers most SQL usage scenarios. Here are some simple examples.

Simple query

Examples:

```
MySQL> SELECT * FROM table1 LIMIT 3;
+--------+----------+----------+------+
| siteid | citycode | username | pv   |
+--------+----------+----------+------+
|      2 |        1 | 'grace'  |    2 |
|      5 |        3 | 'helen'  |    3 |
|      3 |        2 | 'tom'    |    2 |
+--------+----------+----------+------+
5 rows in set (0.01 sec)

MySQL> SELECT * FROM table1 ORDER BY citycode;
+--------+----------+----------+------+
| siteid | citycode | username | pv   |
+--------+----------+----------+------+
|      2 |        1 | 'grace'  |    2 |
|      1 |        1 | 'jim'    |    2 |
|      3 |        2 | 'tom'    |    2 |
|      4 |        3 | 'bush'   |    3 |
|      5 |        3 | 'helen'  |    3 |
+--------+----------+----------+------+
5 rows in set (0.01 sec)
```

Join query

Examples:

```
MySQL> SELECT SUM(table1.pv) FROM table1 JOIN table2 WHERE table1.siteid = table2.siteid;
+------------------+
| sum(`table1`.`pv`) |
+------------------+
|               12 |
+------------------+
1 row in set (0.20 sec)
```

Subquery

Examples:

```
MySQL> SELECT SUM(pv) FROM table2 WHERE siteid IN (SELECT siteid FROM table1 WHERE siteid > 2);
+----------+
| sum(`pv`) |
+----------+
|        8 |
+----------+
1 row in set (0.13 sec)
```

Advanced Operations Guide

Some common advanced features of Palo are listed in this document to help users fully understand Palo.

The specific instructions of some features will be introduced in a separate document.

Relationship table, partition and bucket division

In Palo, user data are stored in a two-dimensional relational table. And based on Shard-Nothing distributed architecture of Palo, the data of a table will be divided into multiple data slices (tablet) horizontally according to the partition and bucket division method specified by the user, and stored on different nodes.

Refer to Relation model and data partition document for specific instructions of partition and bucket division.

Data model

One of the characteristics of Palo is supporting both detailed data query and aggregate data query. The user can specify the data model of the table to adapt to different application scenarios.

Palo currently supports three data models: 1) Duplicate detail model. 2) Aggregate aggregation model. 3) Unqiue primary key model.

Refer to Data model document for specific instructions and usage suggestions for the three models.

Materialized view

Materialized view is a kind of data analysis acceleration technology, and Palo supports creating materialized views based on basic tables. For example, an aggregate view based on partial columns can be established on the table of detail data model, which can satisfy the fast query of detail data and aggregate data at the same time.

Also, Palo can automatically ensure the data consistency of materialized views and basic tables, and automatically match the appropriate materialized views when querying, which greatly reduces the data maintenance cost of users, providing users with a consistent and transparent query acceleration experience.

Refer to [Materialized view]（TODO） document for specific instructions of materialized views.

Changing of table structure

Palo supports online table structure changes, whose operations include adding, deleting, rearranging, modifying column types, adding and deleting partitions, and renaming libraries, tables, and partitions. All these operations will not affect the current loading or query, and can ensure that users can smoothly change the table structures in the production environment.

Refer to [Changing of table structure]（TODO）document for specific instructions of all change operations.

Multiple loading methods

We have introductions about how to load data stored in BOS in Basic operation guides. Besides, Palo itself supports various loading methods, such as loading local data through HTTP protocol or subscribing to messages in Kafka through Route Load function. The data can also be loaded directly through INSERT statement in real time .

Refer to Loading overview for more loading methods.

Data deletion and update

Palo supports two ways to delete loaded data. One is by specifying WHERE condition through DELETE FROM statement to delete data. This method is more general and suitable for timing deletion task with lower frequency.

The other method, only used for Unique primary key model, is to load primary key row data that are needed to be deleted by loading data .Palo The data are physically deleted by deleting the sign bits in Palo internally. This method is suitable for deleting data in real time.

Refer to Data update document for detailed instructions of deleting and update operations.

Relation Model and Data Division

This document mainly introduces the table creation and data partition of Palo, as well as the problems and solutions that may be encountered in table creation.

Basic concepts

In Palo, data are logically described in the form of relation tables.

Row & Column

A table includes rows and columns. Row is a row of data of a user. Column is used to describe different fields in a row of data.

In the default data model, columns are only divided into sorted and non-sorted. The storage engine will sort and store the data according to the sorting sequence, and build a sparse index to quickly search the sorted data.

In the aggregation model, columns can be divided into two categories: Key and Value. Key and Value, from the perspective of business, correspond to dimension column and indicator column respectively. From the perspective of aggregation model, rows with the same key column are aggregated into one row. And the aggregation mode of Value column is specified by the user when creating the table. Refer to Data model document for more instructions of aggregation model.

Partition & Tablet

In storage engine of Palo, user data are first divided into several partitions, and the partition rule is usually based on the partition column specified by the user, such as by time. In each partitions, the data are further divided into buckets by installing Hash, and the rule of bucket is to find the value of bucket column specified by the user for Hash bucket. Each bucket is a data tablet, which is also the smallest logical unit of data partition.

Stored independently, the direct data of tablet have no intersection. Tablet is also the smallest physical storage unit for data movement, copying and other operations.

Partition can be regarded as the smallest management unit in logic. Data can be loaded or deleted for only one partition.

Data partition

We use a table creation operation to illustrate data partition of Palo.

Table creation of Palo is a synchronous command, if the command has successful return, the table is successfully created.

Refer to CREATE TABLE for more help.

This section, through an example, introduces how Palo creates tables.

```
CREATE TABLE IF NOT EXISTS example_db.expamle_tbl
(
    `user_id` LARGEINT NOT NULL COMMENT "user id",
    `date` DATE NOT NULL COMMENT "date and time of data injection",
    `timestamp` DATETIME NOT NULL COMMENT "timestamp of data injection",
    `city` VARCHAR(20) COMMENT "user's city",
    `age` SMALLINT COMMENT "user's age",
    `sex` TINYINT COMMENT "user's sex",
    `last_visit_date` DATETIME REPLACE DEFAULT "1970-01-01 00:00:00" COMMENT "last visit time of user",
    `cost` BIGINT SUM DEFAULT "0" COMMENT "Total consumption of user",
    `max_dwell_time` INT MAX DEFAULT "0" COMMENT "Maximum dwelling time of users",
    `min_dwell_time` INT MIN DEFAULT "99999" COMMENT "Minimum dwelling time of users"
)
ENGINE=olap
AGGREGATE KEY(`user_id`, `date`, `timestamp`, `city`, `age`, `sex`)
PARTITION BY RANGE(`date`)
(
    PARTITION `p201701` VALUES LESS THAN ("2017-02-01"),
    PARTITION `p201702` VALUES LESS THAN ("2017-03-01"),
    PARTITION `p201703` VALUES LESS THAN ("2017-04-01")
)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 16
PROPERTIES
(
    "replication_num" = "3",
    "storage_medium" = "SSD",
    "storage_cooldown_time" = "2018-01-01 12:00:00"
);
```

Column definition

Here we only take AGGREGATE KEY data model as an example. Refer to Data model for more data models.

Refer to CREATE TABLE document for basic types of columns.

In AGGREGATE KEY data model, all columns without specified aggregation methods (SUM, REPLACE, MAX, MIN) are regarded as Key columns. The rest are the Value column.

Refer to the following suggestions for column definition:

1. Key column must precede all Value columns.
2. Choose an integer type as much as possible. Because the calculation and search efficiency of integer type are much higher than those of character strings.
3. For the selection principle of integer types of different lengths, follow **Enough is OK** .
4. For the lengths of VARCHAR and CHAR types, follow **Enough is OK** .

5. The total byte length of all columns (including Key and Value) cannot exceed 100KB.

Partition and bucket

Palo supports two layers of data partition. The first layer is Partition, which only supports Range partition. The second layer is Bucket (Tablet), which only supports Hash partition.

You can also use only one layer of partition. When using a layer of partition, only Bucket is supported.

1. Partition

   - Partition column can specify one or more columns and the partition column must be Key column. The usage of multi column partition is summarized in the following **Multi column partition**.
   - No matter what type of partition column is, double quotes are required when writing partition values.
   - The partition column is usually a time column to facilitate the management of old and new data.
   - In theory, there is no upper limit on the number of partitions.
   - When creating a table without Partition, the system will automatically generate a Partition with the same name as the table name and a full range of values. The Partition is invisible to the user and cannot be deleted or modified.
   - Partition supports specifying only the upper bound through `VALUES LESS THAN (...)`, the system will take the upper bound of the previous partition as the lower bound of the partition and generate a left closed and right open interval. It also supports specifying both upper and lower bounds through `VALUES [...)` to generate a left closed and right open interval.
   - It is easy to understand that `VALUES [...)` is used to specify both upper and lower bounds. Here is an example to show how the partition range changes when the `VALUES LESS THAN (...)` statement is used to add or delete partitions
     - When the table is created, we can see in the above example, the following three partitions will be automatically generated:

```
p201701: [MIN_VALUE, 2017-02-01)
p201702: [2017-02-01, 2017-03-01)
p201703: [2017-03-01, 2017-04-01)
```

     - When we add a partition p201705 VALUES LESS THAN ("2017-06-01") , the partition result is as follows:

```
 p201701: [MIN_VALUE, 2017-02-01)
p201702: [2017-02-01, 2017-03-01)
p201703: [2017-03-01, 2017-04-01)
p201705: [2017-04-01, 2017-06-01)
```

- Now we delete the partition p201703, and the partition result is as follows:

```
 p201701: [MIN_VALUE, 2017-02-01)
p201702: [2017-02-01, 2017-03-01)
p201705: [2017-04-01, 2017-06-01)
```

> We can note that the partition ranges of p201702 and p201705 have not changed, but there is an empty hole between the two partitions: [2017-03-01, 2017-04-01].This means that if the loaded data range is within this empty hole range, the data cannot be loaded.

- We continue to delete partition p201702, and the partition result is as follows:

```
 p201701: [MIN_VALUE, 2017-02-01)
p201705: [2017-04-01, 2017-06-01)
The empty hole range is : [2017-02-01, 2017-04-01)
```

- Now we add a partition p201702new VALUES LESS THAN ("2017-03-01"), and the partition result is as follows:

```
 p201701:    [MIN_VALUE, 2017-02-01)
p201702new: [2017-02-01, 2017-03-01)
p201705:    [2017-04-01, 2017-06-01)
```

> The empty hole range is reduced to: [2017-03-01, 2017-04-01)

- Now we delete the partition p201701 and add the partition p201612 VALUES LESS THAN ("2017-01-01"), the partition result is as follows:

```
p201612:    [MIN_VALUE, 2017-01-01)
p201702new: [2017-02-01, 2017-03-01)
p201705:    [2017-04-01, 2017-06-01)
```

> There occurs a new empty hole: [2017-01-01, 2017-02-01)

To sum up, the deletion of partitions will not change the scope of existing partitions. Deleting a partition may result in an empty hole. When adding a partition through `VALUES LESS THAN` statement, the lower bound of the partition is immediately followed by the upper bound of the previous partition.

Adding partitions with overlapping ranges is not allowed.

2. Bucket
   - If Partition is used, the `DISTRIBUTED ...` statement describes the partition rules of the data in **every partitions**. While if Partition is not used, it describes the partition rules for the data of the whole table.
   - The bucket column can be multiple columns, but they must be Key columns. The bucket column can be the same or different from the partition column.
   - The selection of bucket columns is a trade-off between **query throughput** and **query concurrency**.
     1. If multiple bucket columns are selected, the data distribution is more uniform. If a query condition does not contain the equivalent conditions of all bucket columns, the query will trigger all bucket scanning simultaneously, so the query throughput will increase and the delay of a single query will decrease. This method is suitable for high throughput and low concurrency query scenarios.
     2. If only one or a few bucket columns are selected, the corresponding point query can trigger only one bucket scan. At this time, when multiple point queries are concurrent, these queries are more likely to trigger different bucket scans respectively, and the IO impact between queries is relatively small (especially when different buckets are distributed on different disks), so this method is suitable for high concurrency point query scenarios.
   - Theoretically, there is no upper limit on the number of buckets.
3. Suggestions on the number and data volume of Partition and Bucket.
   - The total number of Tablets in a table is equal to (Partition num * Bucket num).
   - The number of Tablet in a table is recommended to be slightly more than the number of disks in the whole cluster without considering the expansion.

- Theoretically, there is no upper and lower bound for data volume in a single Tablet, but the range of 1G - 10G is recommended. If the data volume of a single Tablet is too small, the data aggregation effect is unfavorable, and the metadata management pressure is high. If the data volume is too large, it is adverse to the migration and completion of the replica, which will increase the cost of Schema Change or Rollup operation failure retrial (the granularity of these operation failure retrial is Tablet).
- When the data volume principle of Tablet conflicts with the quantity principle, data volume principle is recommended priority to quantity principle.
- When creating a table, the number of Bucket in each partitions is specified uniformly. However, when adding partitions dynamically (`ADD PARTITION`), the number of Bucket in the new partition can be specified separately. This function can easily deal with data shrinkage or expansion.
- Once the number of Buckets in a Partition is specified, it is unchangeable. Therefore, when determining the number of Bucket, the cluster expansion should be considered in advance. For example, there are only 3 hosts at present, each host has 1 disk, and if the number of Buckets is set to 3 or less, the concurrency will not be increased even if the machine is added afterwards.
- Here is an example: Suppose there are 10 BE and one disk for each BE, if the total size of a table is 500MB, then 4-8 partitions can be considered. 5GB: 8-16 partitions。50GB: 32 partitions. 500GB: It is recommended to set each partition to 50GB or so, and each partition has 16-32 tablets. 5TB: It is recommended to set each partition to 50GB or so, and each partition has 16-32 tablets.

> Note: the data volume in the table can be viewed through `show data` command, and the result should be divided by the number of copies to get the data volume of the table.

Multi-column partition

Palo supports specifying multiple columns as partition columns. The example is as follows:

```
 PARTITION BY RANGE(`date`, `id`)
(
    PARTITION `p201701_1000` VALUES LESS THAN ("2017-02-01", "1000"),
    PARTITION `p201702_2000` VALUES LESS THAN ("2017-03-01", "2000"),
    PARTITION `p201703_all`  VALUES LESS THAN ("2017-04-01")
)
```

In the above example, we specify `date` (DATE type) and `id` (INT type) as partition columns. The final partition of the above example is as follows:

```
 * p201701_1000:    [(MIN_VALUE,  MIN_VALUE), ("2017-02-01", "1000")   )
* p201702_2000:    [("2017-02-01", "1000"),  ("2017-03-01", "2000")   )
* p201703_all:     [("2017-03-01", "2000"),  ("2017-04-01", MIN_VALUE))
```

Note that in the last partition, the user only specifies the partition column value as`date` by default, so the partition value of `id` column will be filled in by default `MIN_VALUE`. When the user inserts data, the partition column values will be compared in order to get the corresponding partitions. Here are the examples:

```
 * Data  -->  Partition
* 2017-01-01, 200     --> p201701_1000
* 2017-01-01, 2000    --> p201701_1000
* 2017-02-01, 100     --> p201701_1000
* 2017-02-01, 2000    --> p201702_2000
* 2017-02-15, 5000    --> p201702_2000
* 2017-03-01, 2000    --> p201703_all
* 2017-03-10, 1       --> p201703_all
* 2017-04-01, 1000    --> Unable to load
* 2017-05-01, 1000    --> Unable to load
```

Data Model

This document mainly describes the data model of Palo from the logical level to help users better use Palo to deal with different business scenarios.

Basic concepts

In Palo, data are described logically in the form of Table. A table includes rows and columns. A row is a row of data for a user. Column is used to describe different fields in a row of data.

Columns can be divided into two categories: Key and Value. From the perspective of the business, Key and Value can correspond to dimension column and indicator column respectively.

The data models of PALO can be divided into three categories

- Duplicate detail model
- Aggregate aggregation model
- Unique primary key model

Next we will introduce them separately.

Duplicate detail model

The detail model is the default data model used by PALO. The data model does not process the loaded data. The data in the table is the original data loaded by the user.

ColumnName	Type	SortKey	Comment
timestamp	DATETIME	Yes	Log time
type	INT	Yes	Log type
error_code	INT	Yes	Error code
error_msg	VARCHAR(1024)	No	Error details
op_id	BIGINT	No	id of person in charge
op_time	DATETIME	No	Processing time

The table creation statements are as follows:

```
 CREATE TABLE IF NOT EXISTS example_db.expamle_tbl
(
    `timestamp` DATETIME NOT NULL COMMENT "Log time",
    `type` INT NOT NULL COMMENT "Log type",
    `error_code` INT COMMENT "Error code",
    `error_msg` VARCHAR(1024) COMMENT "Error details",
    `op_id` BIGINT COMMENT "id of person in charge",
    `op_time` DATETIME COMMENT "Processing time"
)
DUPLICATE KEY(`timestamp`, `type`)
... /* Omit Partition and Distribution information */
 ;
```

The DUPLICATE KEY specified in the table creation statements is only used to indicate which columns the underlying data are sorted by. （A more appropriate name should be "Sorted Column" , the name "DUPLICATE KEY" is here only used to clearly indicate the data model used. For more explanation of "Sorted Column", refer to Index document.) In selecting DUPLICATE KEY, we suggest that the first 2-4 columns should be selected appropriately.

This data model is suitable for the storage of original data without aggregation requirements and primary key uniqueness constraint. Additionally, users can build aggregation view based on this model through materialized view function, so it is a recommended data model.

Aggregate aggregation model

Aggregation model requires users, when creating tables, to explicitly divide columns into Key columns and Value columns. The model will automatically aggregate the rows with the same Key on the Value column.

Let's illustrate, with practical example, what an aggregation model is and how to use it correctly.

Example 1: loading data aggregation

Suppose we have the following data table modes in the business:

ColumnName	Type	AggregationType	Comment
user_id	LARGEINT		User id
date	DATE		Data injection date
city	VARCHAR(20)		User's city
age	SMALLINT		User's age
sex	TINYINT		User's sex
last_visit_date	DATETIME	REPLACE	Last visit time of user
cost	BIGINT	SUM	Total consumption of user
max_dwell_time	INT	MAX	Maximum dwelling time of user
min_dwell_time	INT	MIN	Minimum dwelling time of user

Convert them to the following table creating statements (omit the partition and distribution information in the table creating statements)

```
CREATE TABLE IF NOT EXISTS example_db.expamle_tbl
(
    `user_id` LARGEINT NOT NULL COMMENT "User id",
    `date` DATE NOT NULL COMMENT "Data injection date",
    `city` VARCHAR(20) COMMENT "User's city",
    `age` SMALLINT COMMENT "User's age",
    `sex` TINYINT COMMENT "User's sex",
    `last_visit_date` DATETIME REPLACE DEFAULT "1970-01-01 00:00:00" COMMENT "Last visit time of user",
    `cost` BIGINT SUM DEFAULT "0" COMMENT "Total consumption of user",
    `max_dwell_time` INT MAX DEFAULT "0" COMMENT "Maximum dwelling time of user",
    `min_dwell_time` INT MIN DEFAULT "99999" COMMENT "Minimum dwelling time of user",
)
AGGREGATE KEY(`user_id`, `date`, `timestamp`, `city`, `age`, `sex`)
... /* Omit Partition and Distribution information */
;
```

As we can see, this is a typical fact table of user information and visit behavior. In general star model, user information and visit behavior are stored in dimension table and fact table respectively. Here, we store the two parts of information in a table for more convenient explanation of PALO data model.

The columns in the table are divided into Key (dimension column) and Value (indicator column) according to whether `AggregationType` is set. Columns without set `AggregationType` such as `user_id`, `date`, `age` are called Key, while columns with set`AggregationType` are called Value.

In loading the data, the same row for the Key column is aggregated into one row, and the Value column is aggregated according to the set `AggregationType`. Currently, there are four ways of aggregation in `AggregationType`:

1. SUM: Sum the Values of multiple rows.
2. REPLACE: replace, the value in the next batch of data will replace the Value in the previously loaded row.

3. MAX: reserve the maximum value.

4. MIN: reserve the minimum value.

Suppose we have the following loaded data (original data): -

user_id	date	city	age	sex	last_visit_date	cost	max_dwell_time	min_dwell_time
10000	2017-10-01	Beijing	20	0	2017-10-01 06:00:00	20	10	10
10000	2017-10-01	Beijing	20	0	2017-10-01 07:00:00	15	2	2
10001	2017-10-01	Beijing	30	1	2017-10-01 17:05:45	2	22	22
10002	2017-10-02	Shanghai	20	1	2017-10-02 12:59:12	200	5	5
10003	2017-10-02	Guangzhou	32	0	2017-10-02 11:20:00	30	11	11
10004	2017-10-01	Shenzhen	35	0	2017-10-01 10:00:15	100	3	3
10004	2017-10-03	Shenzhen	35	0	2017-10-03 10:20:22	11	6	6

Suppose that this is a table that records the behaviors of users visiting a product page. Let's take the first row of data as an example to explain:

Data	Instructions
10000	user id，unique id for each user
2017-10-01	Data storage time, accurate to date
beijing	User's city
20	User's age
0	Male（1 is for female）
2017-10-01 06:00:00	The time that the user visits this page, accurate to seconds
20	Consumption generated by the user's current visit
10	The dwelling time of the user's visiting the page this time
10	The dwelling time of the user's visiting the page this time (redundancy)

Then, load this batch of data correctly into Palo, and the final storage in Palo is as follows:

user_id	date	city	age	sex	last_visit_date	cost	max_dwell_time	min_dwell_time
10000	2017-10-01	Beijing	20	0	2017-10-01 07:00:00	35	10	2
10001	2017-10-01	Beijing	30	1	2017-10-01 17:05:45	2	22	22
10002	2017-10-02	Shanghai	20	1	2017-10-02 12:59:12	200	5	5
10003	2017-10-02	Guangzhou	32	0	2017-10-02 11:20:00	30	11	11
10004	2017-10-01	Shenzhen	35	0	2017-10-01 10:00:15	100	3	3
10004	2017-10-03	Shenzhen	35	0	2017-10-03 10:20:22	11	6	6

We can see that user 10000 has only one row of **aggregated** data left, while the data of other users are consistent with the original data. Here we first explain the aggregated data of user 10000:

There are no changes in the first 5 columns, the change is from column 6 `last_visit_date` :

- `2017-10-01 07:00:00`: Because the aggregation method of the column `last_visit_date` is REPLACE, ``2017-10-01 06:00:00` is replaced by `2017-10-01 07:00:00` and is saved.

> Note: for the data in the same loading batch, the replacement order is not guaranteed for REPLACE aggregation method. Probably `2017-10-01 06:00:00` will be saved finally in this example. For the data in different loading batches, it can be guaranteed that the data in the later batches will replace the data in the previous batches.

- `35`: Because the aggregate type of `cost` column is SUM, 35 is obtained by adding 20 + 15.
- `10`: Because the aggregate type of `max_dwell_time` column is MAX, 10 is obtained by taking the maximum value between 10 and 2 .
- `2`: Because the aggregate type of `min_dwell_time` column is MIN, 2 is obtained by taking the minimum value from 10 and 2.

Only the aggregated data will be stored in Palo after aggregation. In other words, detailed data will be lost and users can no longer query the detailed data before aggregation.

Example 2: reserving detail data

The structure of table, following example 1, is modified as follows:

ColumnName	Type	AggregationType	Comment
user_id	LARGEINT		User id
date	DATE		Data injection date
timestamp	DATETIME		Data injection time, accurate to seconds
city	VARCHAR(20)		User's city
age	SMALLINT		User's age
sex	TINYINT		User's sex
last_visit_date	DATETIME	REPLACE	Last visit time
cost	BIGINT	SUM	Total consumption of user
max_dwell_time	INT	MAX	Maximum dwelling time of user
min_dwell_time	INT	MIN	Minimum dwelling time of user

Now, a column of `timestamp` is added to record the data injection time accurate to seconds.

Loaded data are as follows:

user_id	date	timestamp	city	age	sex	last_visit_date	cost	max_dwell_time	min_dwell_time
10000	2017-10-01	2017-10-01 08:00:05	Beijing	20	0	2017-10-01 06:00:00	20	10	10
10000	2017-10-01	2017-10-01 09:00:05	Beijing	20	0	2017-10-01 07:00:00	15	2	2
10001	2017-10-01	2017-10-01 18:12:10	Beijing	30	1	2017-10-01 17:05:45	2	22	22
10002	2017-10-02	2017-10-02 13:10:00	Shanghai	20	1	2017-10-02 12:59:12	200	5	5
10003	2017-10-02	2017-10-02 13:15:00	Guangzhou	32	0	2017-10-02 11:20:00	30	11	11
10004	2017-10-01	2017-10-01 12:12:48	Shenzhen	35	0	2017-10-01 10:00:15	100	3	3
10004	2017-10-03	2017-10-03 12:38:20	Shenzhen	35	0	2017-10-03 10:20:22	11	6	6

Then, load this batch of data correctly into Palo, and the final storage in Palo is as follows:

user_id	date	timestamp	city	age	sex	last_visit_date	cost	max_dwell_time	min_dwell_time
10000	2017-10-01	2017-10-01 08:00:05	Beijing	20	0	2017-10-01 06:00:00	20	10	10
10000	2017-10-01	2017-10-01 09:00:05	Beijing	20	0	2017-10-01 07:00:00	15	2	2
10001	2017-10-01	2017-10-01 18:12:10	Beijing	30	1	2017-10-01 17:05:45	2	22	22
10002	2017-10-02	2017-10-02 13:10:00	Shanghai	20	1	2017-10-02 12:59:12	200	5	5
10003	2017-10-02	2017-10-02 13:15:00	Guangzhou	32	0	2017-10-02 11:20:00	30	11	11
10004	2017-10-01	2017-10-01 12:12:48	Shenzhen	35	0	2017-10-01 10:00:15	100	3	3
10004	2017-10-03	2017-10-03 12:38:20	Shenzhen	35	0	2017-10-03 10:20:22	11	6	6

We can see that the stored data are exactly the same as the loaded data without any aggregation. This is because the `timestamp` column is added to this batch of data, and the Keys of all rows are **not exactly the same**. In other words, as long as the Key of each row in the loaded data are not exactly the same, Palo can save the complete detail data even in the aggregation model.

Example 3: aggregation of loaded data and existing data

Following example 1, suppose that the data in the table are as follows:

user_id	date	city	age	sex	last_visit_date	cost	max_dwell_time	min_dwell_time
10000	2017-10-01	Beijing	20	0	2017-10-01 07:00:00	35	10	2
10001	2017-10-01	Beijing	30	1	2017-10-01 17:05:45	2	22	22
10002	2017-10-02	Shanghai	20	1	2017-10-02 12:59:12	200	5	5
10003	2017-10-02	Guangzhou	32	0	2017-10-02 11:20:00	30	11	11
10004	2017-10-01	Shenzhen	35	0	2017-10-01 10:00:15	100	3	3
10004	2017-10-03	Shenzhen	35	0	2017-10-03 10:20:22	11	6	6

Let's continue to load a new batch of data:

user_id	date	city	age	sex	last_visit_date	cost	max_dwell_time	min_dwell_time
10004	2017-10-03	Shenzhen	35	0	2017-10-03 11:22:00	44	19	19
10005	2017-10-03	Changsha	29	1	2017-10-03 18:11:02	3	1	1

Then, load this batch of data correctly into PALO, and the final storage in PALO is as follows:

user_id	date	city	age	sex	last_visit_date	cost	max_dwell_time	min_dwell_time
10000	2017-10-01	Beijing	20	0	2017-10-01 07:00:00	35	10	2
10001	2017-10-01	Beijing	30	1	2017-10-01 17:05:45	2	22	22
10002	2017-10-02	Shanghai	20	1	2017-10-02 12:59:12	200	5	5
10003	2017-10-02	Guangzhou	32	0	2017-10-02 11:20:00	30	11	11
10004	2017-10-01	Shenzhen	35	0	2017-10-01 10:00:15	100	3	3
10004	2017-10-03	Shenzhen	35	0	2017-10-03 11:22:00	55	19	6
10005	2017-10-03	Changsha	29	1	2017-10-03 18:11:02	3	1	1

It can be seen that the existing data of user 10004 and the newly-loaded data are aggregated with the data of the user 10005 added.

There are three stages of data aggregation in PALO:

1. ETL stage of each batch data loading. This stage aggregates the data loaded in each batch.
2. The stage when the underlying BE performs data Comparison. In this stage, BE will further aggregate the loaded data of different batches.
3. Data query stage. In data query, the data involved in the query will be aggregated.

The degree of data aggregation may be inconsistent at different times. When a batch of data are loaded, for example, they may not have been aggregated with the existing data. But for users, users **can only query** aggregated data, which means that different aggregation degrees are transparent for user queries. Users should always think that the data exists with **the degree of final aggregation**, and **should not assume that some aggregation has not occurred**. (Please refer to the section of **limitations of aggregation model** for more details.)

Unique primary key model

Users, in some multi-dimensional analysis scenarios, pay more attention to how to ensure the uniqueness of the Key, that is, how to obtain the uniqueness constraint of the Primary Key. Therefore, we introduce Unique data model, which is, in essence, a special case of aggregation model and a simplified table structure representation. Here are the examples:

ColumnName	Type	IsKey	Comment
user_id	BIGINT	Yes	User id
username	VARCHAR(50)	Yes	User's nickname
city	VARCHAR(20)	No	User's city
age	SMALLINT	No	User's age
sex	TINYINT	No	User's sex
phone	LARGEINT	No	User's phone number
address	VARCHAR(500)	No	User's address
register_time	DATETIME	No	User registration time

This is a typical table of user base information. There is no aggregation requirement for this kind of data, and they just need to ensure the uniqueness of the primary key.（The primary key here is user_id + username）.Then our table creation statements are as follows:

```
 CREATE TABLE IF NOT EXISTS example_db.expamle_tbl
(
    `user_id` LARGEINT NOT NULL COMMENT " User id",
    `username` VARCHAR(50) NOT NULL COMMENT "User's nickname",
    `city` VARCHAR(20) COMMENT "User's city",
    `age` SMALLINT COMMENT "User's age",
    `sex` TINYINT COMMENT "User's sex",
    `phone` LARGEINT COMMENT "User's phone number",
    `address` VARCHAR(500) COMMENT "User's address",
    `register_time` DATETIME COMMENT "User registration time "
)
UNIQUE KEY(`user_id`, `user_name`)
... /* Omit Partition and Distribution information */
 ;
```

This table structure is exactly the same as the following table structure described by the aggregation model:

ColumnName	Type	AggregationType	Comment
user_id	BIGINT		User id
username	VARCHAR(50)		User's nickname
city	VARCHAR(20)	REPLACE	User's city
age	SMALLINT	REPLACE	User's age
sex	TINYINT	REPLACE	User's sex
phone	LARGEINT	REPLACE	User's phone number
address	VARCHAR(500)	REPLACE	User's address
register_time	DATETIME	REPLACE	User registration time

And the table creation statements are as follows:

```
 CREATE TABLE IF NOT EXISTS example_db.expamle_tbl
(
    `user_id` LARGEINT NOT NULL COMMENT "User id",
    `username` VARCHAR(50) NOT NULL COMMENT "User's nickname",
    `city` VARCHAR(20) REPLACE COMMENT "User's city",
    `age` SMALLINT REPLACE COMMENT "User's age",
    `sex` TINYINT REPLACE COMMENT "User's sex",
    `phone` LARGEINT REPLACE COMMENT "User's phone number",
    `address` VARCHAR(500) REPLACE COMMENT "User's address",
    `register_time` DATETIME REPLACE COMMENT "User registration time"
)
AGGREGATE KEY(`user_id`, `user_name`)
... /* Omit Partition and Distribution information */
 ;
```

That is, Unique model can be fully replaced by REPLACE in the aggregation model. Its internal implementation and data storage are exactly the same. No examples here will be given.

Limitations of aggregation model

Aggregation model (including Unique model) , through a precomputing method, can reduce data volume that need to be calculated in real time and speed up the query. The model, however, has limitations in use.

In the aggregation model, the model displays data **after final aggregation** for external presentation. In other words, any data that have not yet been aggregated (for example, data from two different loaded batches) must be displayed in a certain way to ensure consistency. Here are the examples:

Suppose the table structure is as follows:

ColumnName	Type	AggregationType	Comment
user_id	LARGEINT		User id
date	DATE		Data injection date
cost	BIGINT	SUM	Total user consumption

Suppose that the storage engine has the following two batches of data that have been loaded:

batch 1

user_id	date	cost
10001	2017-11-20	50
10002	2017-11-21	39

batch 2

user_id	date	cost
10001	2017-11-20	1
10001	2017-11-21	5
10003	2017-11-22	22

We can see that the data of user 10001 in the two loading batches have not been aggregated. However, in order to ensure that users can only query the final aggregated data as follows:

user_id	date	cost
10001	2017-11-20	51
10001	2017-11-21	5
10002	2017-11-21	39
10003	2017-11-22	22

We add aggregation operator to query engine to ensure the external consistency of data.

In addition, on the aggregate column (Value), when executing an aggregate class query that is inconsistent with the aggregate type, semantics should be paid attention to. For example, we execute the following query in the above example:

```
SELECT MIN(cost) FROM table;
```

The result is 5, not 1.

At the same time, this consistency guarantee will greatly reduce the query efficiency in some queries.

Let's take the most basic count () query as an example:

```
SELECT COUNT(*) FROM table;
```

In other databases, this kind of query will return results quickly. Because in the implementation, we can "count rows and save count statistics" when loading, or we can "scan only a column of data to get count value" when querying, we can get the query results with little overhead. However, in Palo's aggregation model, the overhead of this kind of query is **rather large.**

Let's take the above data as an example

batch 1

user_id	date	cost
10001	2017-11-20	50
10002	2017-11-21	39

batch 2

user_id	date	cost
10001	2017-11-20	1
10001	2017-11-21	5
10003	2017-11-22	22

Because the final aggregation result is:

user_id	date	cost
10001	2017-11-20	51
10001	2017-11-21	5
10002	2017-11-21	39
10003	2017-11-22	22

The correct result of `select count(*) from table;` should be **4**. But if we just scan `user_id`column and add query aggregation to this column, the final result is **3** (10001, 10002, 10003). If the query aggregation is not added, the result is **5** (five rows of data in two batches). It can be seen that both results are incorrect.

In order to get the correct results, we must read the values of columns user_id and date simultaneously, plus aggregation during query, then the correct result of 4 can be returned . In other words, in the count(*) query, Palo must scan all the AGGREGATE KEY columns (here are user_id and date), and only after aggregation can we get the semantic correct results. When there are too many aggregate columns, the count () query needs to scan a large amount of data.

Therefore, when there are frequent count (\ *) queries in the business, we suggest that users simulate **count(*)** by adding a column whose **value is always 1 and aggregate type is SUM**. As for the table structure in the example just now, we modify it as follows:

ColumnName	Type	AggregateType	Comment
user_id	BIGINT		User id
date	DATE		Data injection date
cost	BIGINT	SUM	Total user consumption
count	BIGINT	SUM	Used to count

Add a count column and load the data, the value of the column is **always 1**. Then the result of `select count(*) from table;` is equivalent to `select sum(count) from table;`. But the query efficiency of the latter is much higher than that of the former. However, this method also has limitations, that is, users need to guarantee that they will not repeatedly load rows with the same AGGREGATE KEY columns. Otherwise, `select sum (count) from table;` can only express the number of original loaded rows, not the semantics of `select count(*) from table;`.

Another way is **to change the aggregation type of the above `count` column to REPLACE, and the value is still always 1**. Then the results of `select sum(count) from table;'` and `select count(*) from table;` will be consistent. And in this way, there is no restriction on loading duplicate rows.

Duplicate model

Duplicate model does not have this limitation of the aggregate model. Because the model does not involve aggregation semantics, when doing count () query, you can get the result with correct semantics by selecting any query column.

Suggestions on the selection of data models

Because the data model has been determined at the time of table creation and**cannot be modified**. Therefore, it is **very important** to choose an appropriate data model.

1. Aggregate model can greatly reduce the data volume to be scanned and the amount of query computation by pre-aggregation, which is very suitable for the report query scenarios with fixed patterns. However, the model is not friendly to count (*) query. Also, because the aggregation method on the Value column is fixed, semantic correctness should be considered when other types of aggregation queries are carried out.

2. Unique model can guarantee the uniqueness constraint of the primary key for the scenarios that need the unique primary key constraint. But we can't take advantage of the query advantage brought by Rollup and other pre-aggregation (because the essence is REPLACE and there is no SUM aggregation method).

3. Duplicate is suitable for Ad-hoc queries of any dimension. Although it is also unable to take advantage of the pre-aggregation feature, it is not constrained by the aggregation model, and can take advantage of the column storage model (only read related columns, but not all Key columns).

Index

Index is used to help filter or find data quickly.

Currently, Palo mainly supports two kinds of indexes: built-in intelligent index, including prefix index and ZoneMap index. Secondary indexes created by users include Bloom Filter index and Bitmap inverted index.

ZoneMap index is the index information automatically maintained for each columns in column storage format, including Min / Max, number of Null values, etc. This kind of index is transparent to users and is not described here. The following mainly introduces the other three types of indexes.

Prefix index

Principles

In essence, Palo's data are stored in a data structure similar to SStable (Sorted String Table), which is an ordered data structure and can be sorted and stored according to the specified columns. In this data structure, it will be very efficient to search by arranging sequence.

The prefix index, which is based on sorting, is an index method to query data quickly according to the given prefix columns.

Prefix index is a sparse index created with the granularity of Block, a block contains 1024 rows of data, and for each Block, the value of the prefix column of the first row of data of the Block is used as the index.

We use the first **36 bytes** of a row of data as the prefix index of this row of data. When a VARCHAR type is encountered, the prefix index is truncated directly. Here are the examples:

1. The prefix index for the following table structure is user_id(8Byte) + age(4Bytes) + message(prefix 20 Bytes).

ColumnName	Type
user_id	BIGINT
age	INT
message	VARCHAR(100)
max_dwell_time	DATETIME
min_dwell_time	DATETIME

2. The prefix index for the following table structure is user user_name(20 Bytes). Because VARCHAR is encountered, it will be truncated directly and will not continue in the future even if it does not reach 36 bytes.

ColumnName	Type
user_name	VARCHAR(20)
age	INT
message	VARCHAR(100)
max_dwell_time	DATETIME
min_dwell_time	DATETIME

Usage scenarios

When our query condition is the **prefix of prefix index**, the query speed can be greatly increased. Now we execute the following

query in the first example:

`SELECT * FROM table WHERE user_id=1829239 and age=20 ;`

The efficiency of this query is **much higher than** that of the following query:

`SELECT * FROM table WHERE age=20 ;`

Therefore, when creating a table, **the correct selection of column order can greatly improve the query efficiency.**

Bloom Filter index

Principles

When creating a table, users can specify to create Bloom Filter index on some columns (hereinafter referred to as BF index). We can also add a BF index by using `ALTER TABLE` command at run time.

Bloom Filter is essentially a bitmap structure used to quickly determine whether a given value is in a set. This kind of judgment will produce a small probability of misjudgment. That is, if `False` is returned, it must not be in this set. And if the range is `True`, it may be in this set.

BF index is also created for granularity by Block. In each Block, the value of the specified column is used as a set to generate a BF index entry, which is used to quickly filter the data that does not meet the conditions in the query.

Applicable scenarios

Due to the characteristics of Bloom Filter data structure, BF index is more suitable to be created on columns with high cardinality, such as UserID. If it is created on a low cardinality column, such as the "sex"column, then every Block will contain almost all values, making BF index meaningless.

Bitmap index

Principles

When creating a table, users can specify to create a Bitmap index on some columns. Also, users can add a new Bitmap index by ALTER TABLE command at run time.

Bitmap index is a special database index technology, which uses bit array (or bitmap, bit set, bit string, bit vector) to store and calculate. Each bit in the position code indicates the existence of the data row corresponding to the key value. A bitmap may point to dozens or even hundreds of rows of data.

Compared with the B * tree index, this way of storing data takes up very little space and is very fast to create and use. When querying according to the key value, users can quickly locate the specific row number according to Bitmap index. When we do and / or or in (x, y,..) query according to the key value, we directly do so or operate through the bitmap of the index to quickly get the result row data.

There are the following restrictions for Bitmap index in Palo:

- Bitmap index is created only on a single column
- The data types supported by bitmap index are as follows:
    - `TINYINT`
    - `SMALLINT`
    - `INT`
    - `UNSIGNEDINT`
    - `BIGINT`

- `CHAR`
- `VARCHAE`
- `DATE`
- `DATETIME`
- `LARGEINT`
- `DECIMAL`
- `BOOL`

Applicable scenarios

1. It is suitable for columns with low cardinality and It is recommended to be between 100 and 100000, such as occupation, city, etc. If the cardinality is too high, there is no obvious advantage; If the cardinality is too low, the space efficiency and performance will be greatly reduced.
2. For certain types of queries such as count, or, and as well as other logical operations, only bit operations are needed. Through multiple condition combination query scenarios like `select count(*) from table where city = 'beijing' and job = 'teacher'`, if a bitmap index is established on each query condition column, efficient bit operation can be performed to accurately locate the required data and scan the data. The smaller the filtered result set is, the more obvious the advantage of bitmap index will be.

Operating Manual

Data Load

Load Overview

Supported data sources

Palo provides a variety of data load schemes to choose from for different data sources.

Data sources	load methods
Baidu Object Storage (BOS), HDFS, AFS	Load data with Broker Load
Local file	Load local data
Baidu News Service (Kafka)	Subscribe to Kafka log
MySQL, Oracle, PostgreSQL	Synchronize data through external table
load data through JDBC	Synchronize data throughJDBC
load data in JSON format	Instructions of importing data in JSON format
MySQL binlog	Please wait and see

General description of data load

The following are the instructions for common features of data load for Palo for users to better use the function.

Atomicity guarantee

Every load job in Palo is a complete transaction operation, whether through Broker Load for batch load or through INSERT statement for single import. The load transaction can ensure that the data atoms in a batch take effect instead of partial data write.

Also, each load job has a Label, which is unique under a Database and is used to uniquely identify an load job. Labels can be

specified by users, and partial load functions can be generated automatically by the system.

Label is used to ensure that the corresponding load job can be successfully imported only once. A successfully imported label will be rejected and an error `Label already used` will be reported if it is used again. `At-Most-Once` semantic can be done in Palo through this mechanism. Combined with `At-Least-Once` semantic of the upstream system, `Exactly-Once` semantic of the imported data can be realized.

Refer to load transaction and atomicity for best practices on atomicity guarantee.

Synchronous load and asynchronous import

load methods include synchronous one and asynchronous one. For synchronous import, the return result indicates the success or failure of the import. For asynchronous import, successful return simply means successful operation submission rather than successful data load, the user need to view the running status of the load job through corresponding commands.

Supported data formats

Supported data formats slightly differ in terms of different load methods.

Load methods	Supported formats
Broker Load	Parquet, ORC, csv, gzip
Stream Load	csv, gzip, json
Routine Load	csv, json

Load Local Data

Stream Load aims to load local files into Palo.

Stream Load, unlike other command submission methods, connects and interacts with Palo through HTTP protocol.

HOST: PORT involved in the method should be HTTP protocol port.

- Public cloud users must use the HTTP protocol port of Compute Node (BE), which is 8040 by default.
- Private deployment users can use the HTTP protocol port of Leader Node (FE), which is 8030 by default. However, it is necessary to ensure that the machine network where the client is located can connect the machine where the Compute Node is located.

We use curl command as an example to demonstrate how to Load data in this document.

We give a code example of importing data using Java at the end of the document.

Load data

Stream Load request body is as follows:

```
PUT /api/{db}/{table}/_stream_load
```

1. Create a table

   Create a table to store the data to be imported through `CREATE TABLE` command. Refer to command manual CREATE TABLE for specific Load methods. Examples are as follows：

```
CREATE TABLE IF NOT EXISTS load_test
(
    id INT,
    name VARCHAR(128)
)
DISTRIBUTED BY HASH(id) BUCKETS 8;
```

2. Load data

   Execute the following curl command to Load local file:

```
curl -u user:passwd -H "label:example_label_1" -T /path/to/local/your_file.txt
http://host:port/api/example_db/load_test/_stream_load
```

   - user: passwd is the user created in Palo. The initial user is admin, and the password is set when creating Palo cluster.
   - host: port is the HTTP protocol port of Compute Node, which is 8040 by default, the user can view it on details page of the Intelligent Cloud Palo cluster .
   - label: can be specified in Header to uniquely identify the Load task

   Refer to Stream Load command file for more advanced operation about Stream Load command.

3. Waiting for Load results

   Stream Load command is a synchronization command, the successful return results means successful import. If the Load data is large, it may take a long time to wait. Examples are as follows:

```
{
    "TxnId": 1003,
    "Label": "example_label_1",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 1000000,
    "NumberLoadedRows": 1000000,
    "NumberFilteredRows": 1,
    "NumberUnselectedRows": 0,
    "LoadBytes": 40888898,
    "LoadTimeMs": 2144,
    "BeginTxnTimeMs": 1,
    "StreamLoadPutTimeMs": 2,
    "ReadDataTimeMs": 325,
    "WriteDataTimeMs": 1933,
    "CommitAndPublishTimeMs": 106,
    "ErrorURL": "http://192.168.1.1:8042/api/_load_error_log?
file=__shard_0/error_log_insert_stmt_db18266d4d9b4ee5-
abb00ddd64bdf005_db18266d4d9b4ee5_abb00ddd64bdf005"
}
```


- If the status of `Status` field is `Success`, it indicates that the load is successful.
- Refer to Stream Load command file for detailed instructions of other fields.

Suggestions for use

- Stream Load can only load local files.
- It is suggested that the data volume of an Load request should be controlled within 1 GB. If there are a large number of local files, they can be submitted concurrently in batches.

Java code examples

Here is a simple JAVA example to execute Stream Load:

```
package demo.palo;

load com.google.gson.Gson;
load com.google.gson.reflect.TypeToken;

load java.io.BufferedInputStream;
load java.io.BufferedOutputStream;
load java.io.BufferedReader;
load java.io.File;
load java.io.FileInputStream;
load java.io.InputStream;
load java.io.InputStreamReader;
load java.lang.reflect.Type;
load java.net.HttpURLConnection;
load java.net.URL;
```

```
load java.nio.charset.StandardCharsets;
load java.util.Base64;

public class PaloStreamLoadDemo {
    private final static String HOST = "127.0.0.1"; // Compute Node host
    private final static int PORT = 8040;   // Compute Node HTTP port

    private static final String STREAM_LOAD_URL_PATTERN = "http://%s:%d/api/%s/%s/_stream_load";

    private final static String DB = "example_db";
    private final static String TABLE = "example_tbl";
    private final static String USER = "user";
    private final static String PASSWD = "passwd";
    // local file to be loaded
    private final static String LOAD_FILE = "./data.txt";

    public static void main(String[] args) throws Exception {
        streamLoad();
    }

    private static void streamLoad() throws Exception {
        String loadUrlStr = String.format(STREAM_LOAD_URL_PATTERN, HOST, PORT, DB, TABLE);
        URL loadUrl = new URL(loadUrlStr);
        HttpURLConnection conn = (HttpURLConnection) loadUrl.openConnection();
        conn.setRequestMethod("PUT");
        String auth = String.format("%s:%s", USER, PASSWD);
        String authEncoding = Base64.getEncoder().encodeToString(auth.getBytes(StandardCharsets.UTF_8));
        conn.setRequestProperty("Authorization", "Basic " + authEncoding);
        conn.addRequestProperty("Expect", "100-continue");
        conn.addRequestProperty("Content-Type", "text/plain; charset=UTF-8");
        // set header.
        // your add add any other headers here.
        conn.addRequestProperty("column_separator", ",");
        conn.addRequestProperty("label", "example_label");
        conn.setDoOutput(true);
        conn.setDoInput(true);

        // read and send file content
        File loadFile = new File(LOAD_FILE);
        try (BufferedOutputStream bos = new BufferedOutputStream(conn.getOutputStream());
             BufferedInputStream bis = new BufferedInputStream(new FileInputStream(loadFile));) {
            int i;
            while ((i = bis.read()) > 0) {
                bos.write(i);
            }
        }

        // get response
        int status = conn.getResponseCode();
        String respMsg = conn.getResponseMessage();
        System.out.println("get status: " + status + ", response msg: " + respMsg);

        // parse the response json
        InputStream stream = (InputStream) conn.getContent();
        BufferedReader br = new BufferedReader(new InputStreamReader(stream));
        StringBuilder sb = new StringBuilder();
        String line;
        while ((line = br.readLine()) != null) {
            sb.append(line);
        }
        Type type = new TypeToken<SubmitResult>() {
        }.getType();
        SubmitResult result = new Gson().fromJson(sb.toString(), type);
```

```
        SubmitResult result = new Gson().fromJson(sb.toString(), type);

        System.out.println("Get result status: " + result.Status);
    }

    // The response json class
    public static class SubmitResult {
        public String TxnId;
        public String Label;
        public String Status;
        public String ExistingJobStatus;
        public String Message;
        public String NumberTotalRows;
        public String NumberLoadedRows;
        public String NumberFilteredRows;
        public String NumberUnselectedRows;
        public String LoadBytes;
        public String LoadTimeMs;
        public String BeginTxnTimeMs;
        public String StreamLoadPutTimeMs;
        public String ReadDataTimeMs;
        public String WriteDataTimeMs;
        public String CommitAndPublishTimeMs;
        public String ErrorURL;
    }
}
```

Load Data in BOS

This document mainly teaches how to load data stored in BOS.

Preparations

Please first store the data to be loaded into Palo on Baidu Object Storage (BOS) through the following steps.

1. Open BOS service

   Refer to Start to use BOS

2. Create Bucket

   Refer to Create Bucket

   Note: the domain of the Bucket must be the same as that of the Palo cluster. The Palo domain can usually be viewed in the upper left corner of the Palo console page

3. Upload files to Bucket

   There are two ways to upload files to Bucket.

   Upload files directly through the console, pleaser refer to document Upload Object.

   Upload files via command line tool:

   1. First [Download BOS CLI command line tool](https://cloud.baidu.com/doc/BOS/s/Ejwvyqobd#bos-cliDownload Address#). Here take Linux operation system bce-cli-0.10.10.zip as an example.

   2. Execute the following command to configure BOS CLI after decompression:

```
./bce -c
BOS Access Key ID []: 353b8dexxxxxxxxxb156d3
BOS Secret Access Key []: ea15a18xxxxx29f78e8d77
BCE Security Token [None]:
Default region name [bj]:
Default domain [bj.bcebos.com]:
Default use auto switch domain [yes]:
Default breakpoint_file_expiration [7] days:
Default use https protocol [no]:
Default multi upload thread num [10]:
```

- BOS Access Key ID and BOS Secret Access Key can be obtained by clicking `Account profile -> Security authentication` on the top right corner of the public cloud page.
- Fill in the abbreviations of domain of Bucket in Default region name and Default domain, please refer to [Domain Access](https://cloud.baidu.com/doc/BOS/s/Ck1rk80hn#Access Domain Name (endpoint)#) .
- Other configurations can be configured by default.

3. Upload the file with the following command:

```
./bce bos cp /path/to/local/your_file.txt bos:/your_bucket_name
```

Start load

Palo supports the following two ways to load BOS data.

Submit load job through Broker Load command

Broker, a stateless process service, has been built into Palo cluster and is mainly used to read and write files from external data sources. Broker Load is used to access source data and load data with Broker service.

1. Create a table

   Create a table to store the data to be loaded through `CREATE TABLE` command. Refer to command manual CREATE TABLE for specific load methods. Examples are as follows:

```
CREATE TABLE IF NOT EXISTS load_test
(
    id INT,
    name VARCHAR(128)
)
DISTRIBUTED BY HASH(id) BUCKETS 8;
```

2. Submit Broker Load load job

   Refer to command manual Broker Load for detailed Broker Load command syntax. Examples are as follows:

```
LOAD LABEL example_db.exmpale_label_1
(
    DATA INFILE("bos://your_bucket_name/your_file.txt")
    INTO TABLE load_test
    COLUMNS TERMINATED BY ","
)
WITH BROKER "bos"
(
    "bos_endpoint" = "http://bj.bcebos.com",
    "bos_accesskey" = "353b8dexxxxxxxxxb156d3",
    "bos_secret_accesskey" = "ea15a18xxxxx29f78e8d77"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

   - LABEL: Each load job needs a specified and unique Label, which can be used to view the running status of the load job.
   - WITH BROKER "bos": "bos" is just the Broker service process name and does not represent the data source to be accessed. Broker name can be viewed through `SHOW BROKER` command after connecting to Palo with the user of admin.
   - "bos_accesskey" and "bos_secret_accesskey" can be obtained by clicking `Account profile -> Security authentication` on the top right corner of the public cloud page.
   - "bos_endpoint" is related to the domain of BOS Bucket, refer to [Domain Access] (https://cloud.baidu.com/doc/BOS/s/Ck1rk80hn#Access Domain Name (endpoint)#) to obtain.

3. View load job status

Broker Load is an asynchronous command, successful execution of the command in the second step only means successful submission of the job. Check the following commands to see specific execution.

```
mysql> SHOW LOAD FROM example_db WHERE LABEL="exmpale_label_1"
*************************** 1. row ***************************
         JobId: 10041
         Label: exmpale_label_1
         State: FINISHED
      Progress: ETL:100%; LOAD:100%
          Type: BROKER
       EtlInfo: unselected.rows=0; dpp.abnorm.ALL=0; dpp.norm.ALL=100000000
      TaskInfo: cluster:N/A; timeout(s):14400; max_filter_ratio:0.0
      ErrorMsg: NULL
    CreateTime: 2020-11-17 09:38:04
  EtlStartTime: 2020-11-17 09:38:09
 EtlFinishTime: 2020-11-17 09:38:09
 LoadStartTime: 2020-11-17 09:38:09
LoadFinishTime: 2020-11-17 09:42:07
           URL: N/A
    JobDetails: {"Unfinished backends":{},"ScannedRows":0,"TaskNumber":0,"All backends":
{},"FileNumber":0,"FileSize":0}
1 row in set (0.01 sec)
```

If the status of the `State` field is `FINISHED`, the load is successful and the data can be queried. Refer to `SHOW LOAD` command file for a specific description of the `SHOW LOAD` return result.

4. Cancel load job

   The following commands can cancel a running Broker Load load job:

```
CANCEL LOAD WHERE LABEL="exmpale_label_1";
```

All loaded data will be rolled back after successful cancellation. Palo will automatically ensure that the data atom in an load job takes effect.

> Note: Refer to BROKER LOAD command file for more detailed and advanced functions of the Broker Load command.

load through external table

Also, Palo supports creating a Broker external table to refer to the data stored in BOS, and then loading the data through

INSERT INTO SELECT.

1. Create a table

   Create a table for data storage. Same as above and there is no need to repeat.

2. Create Broker external table

   Refer to CREATE EXTERNAL TABLE command manual for specific instructions for the command of creating external table. Examples are as follows:

```
CREATE EXTERNAL TABLE IF NOT EXISTS example_db.example_ext_table
(
    id INT,
    name VARCHAR(128)
)
ENGINE=BROKER
PROPERTIES
(
    "broker_name" = "bos",
    "path" = "bos://your_bucket_name/your_file.txt",
)
BROKER PROPERTIES
(
    "bos_endpoint" = "http://bj.bcebos.com",
    "bos_accesskey" = "353b8dexxxxxxxxxb156d3",
    "bos_secret_accesskey" = "ea15a18xxxxx29f78e8d77"
);
```

   - ENGINE: The type of ENGINE is BROKER, which means that it is an external table that accesses data with Broker service.
   - "broker_name" is "bos" , "bos" is just the Broker service process name and does not represent the data source to be accessed. Broker name can be viewed through SHOW BROKER command after connecting to Palo with the user of admin.
   - "bos_accesskey" and "bos_secret_accesskey" can be obtained by clicking Account profile -> Security authentication on the top right corner of the public cloud page.
   - "bos_endpoint" is related to the domain of BOS Bucket, refer to [Domain Access] (https://cloud.baidu.com/doc/BOS/s/Ck1rk80hn#Access Domain Name（endpoint）#) to obtain.

> Note: external table data can also be queried directly through SELECT with low efficiency, so it is recommended to execute the query after loading them into Palo.

3. load data

   load data from an external table to an internal table with the following command.

   sql
   INSERT INTO load_test SELECT * FROM example_ext_table;

The command is a synchronous command (the operation of submitting INSERT job asynchronously is under development), the successful command return result means that the data load is completed. When the loaded data amount is large, the task may be canceled due to query timeout. Refer to variable `query_timeout` in Varialbe file for the setting of query timeout.

> Note: Refer to INSERT command file for more detailed and advanced functions of INSERT command.

Subscribe Kafka Log

Through submitting routine load job, the user can directly subscribe to Kafka message data to synchronize the data in near real time.

Palo itself can guarantee to subscribe to Kafka messages without losing or duplication, that is, `Exactly-Once` consumption semantics.

Preparations

Open Baidu Message Service

Please first open Baidu Message Service (BMS), which is based on Kafka to provide hosting services in Baidu intelligent cloud, according to the following process.

1. Open message service according to BMS Quick start file.
2. Download and unzip certificate zip file kafka-key.zip to get the following files
   - ca.pem
   - client.key
   - client.keystore.jks
   - client.pem
   - client.properties
   - client.truststore.jks
3. Upload certificate file to HTTP server.

   Since Palo needs to download these integers from a HTTP server to get access to Kafka later, first we need to upload these certificates to the HTTP server. This HTTP server must can be accessed by the Leader Node of Palo.

   If you don't have an appropriate HTTP server, you can complete it through the following ways by virtue of Baidu Object Storage (BOS):

   1. Open BOS service and create a Bucket according to files Start to use , Create Bucket. **Note that the domain of the Bucket must be the same as that of the Palo cluster.**
   2. Upload the following 3 files to Bucket
      - ca.pem
      - client.key
      - client.pem
   3. Click `File information` on the right side of the file on BOS Bucket file list page to obtain HTTP access connection. Set `Valid connection time` to `-1`, that is, Permanent.

> Note: Do not use the http download address with cdn acceleration, which cannot be accessed by Palo in some cases.

Self-built Kafka service

Make sure that the Kafka service and Palo cluster are in the same VPC and that the networks between them can be interconnected in order to use self-built Kafka service.

Subscribe to Kafka message

Routine Load function in Palo is used to subscribe to Kafka message.

First create a **Routine Load Job**. The job will constantly send a series of**tasks** through routine scheduling, and each task will consume a certain number of messages in Kafka.

Please note the following restrictions for use:

1. Kafka access without authentication and Kafka cluster with SSL authentication are supported.
2. The supported message formats are as follows:
   - csv text format. Each message is set in a line and it doesn't contain **line break** at the end of the line.
   - Json format , refer to Load data in Json format.
3. Only Kafka 0.10.0.0 or above is supported.

Access Kafka cluster with SSL authentication

Routine load function supports Kafka cluster without authentication and Kafka cluster with SSL authentication.

Accessing SSL certified Kafka cluster requires user provide the certificate file for certifying Kafka Broker public key (ca.pem). Public key (client.pem), key file(client.key) and the key password of the client should also be provided if Kafka cluster also opens client authentication. The required files here need to be uploaded to Plao through command `CREAE FILE` and the catalog name should be `kafka`. Refer to CREATE FILE command manual for specific command help. Examples are as follows:

- Upload files

```
CREATE FILE "ca.pem" PROPERTIES("url" = "https://example_url/kafka-key/ca.pem", "catalog" = "kafka");
CREATE FILE "client.key" PROPERTIES("url" = "https://example_urlkafka-key/client.key", "catalog" = "kafka");
CREATE FILE "client.pem" PROPERTIES("url" = "https://example_url/kafka-key/client.pem", "catalog" = "kafka");
```

View the uploaded files through `SHOW FILES` command after upload.

Create a routine load job

Refer to ROUTINE LOAD command manual for specific commands for creating routine load jobs. Examples are as follows:

1. Access Kafka cluster without authentication

```
CREATE ROUTINE LOAD example_db.my_first_job ON example_tbl
COLUMNS TERMINATED BY ","
PROPERTIES
(
    "max_batch_interval" = "20",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200",
)
FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic",
    "property.group.id" = "xxx",
    "property.client.id" = "xxx",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

- `max_batch_interval/max_batch_rows/max_batch_size` is used to control the run cycle of a subtask. The longest running time, the largest number of consumption lines and the largest amount of consumption data determine the running cycle of a subtask.

2. Access Kafka cluster with SSL authentication

```
CREATE ROUTINE LOAD example_db.my_first_job ON example_tbl
COLUMNS TERMINATED BY ",",
PROPERTIES
(
    "max_batch_interval" = "20",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200",
)
FROM KAFKA
(
   "kafka_broker_list"= "broker1:9091,broker2:9091",
   "kafka_topic" = "my_topic",
   "property.security.protocol" = "ssl",
   "property.ssl.ca.location" = "FILE:ca.pem",
   "property.ssl.certificate.location" = "FILE:client.pem",
   "property.ssl.key.location" = "FILE:client.key",
   "property.ssl.key.password" = "abcdefg"
);
```

- For Baidu Message Service, `property.ssl.key.password` property can be obtained from file `client.properties`.

View loaded job status

Refer to SHOW ROUTINE LOAD command file for specific commands and examples of job status.

Refer to SHOW ROUTINE LOAD TASK command file for specific commands and examples of running status of a certain job task.

Only running task can be viewed instead of those that have ended or not started.

Alter job properties

Partial created job properties can be altered. Refer to ALTER ROUTINE LOAD command manual for specific instructions.

Job control

The user can stop, pause and resume the job through three commands: `STOP/PAUSE/RESUME`.

Refer to command files STOP ROUTINE LOAD,PAUSE ROUTINE LOAD , RESUME ROUTINE LOAD for specific commands.

More help

Refer to ROUTINE LOAD command manual for more detailed syntax and best practices of Route Load.

Use JDBC to Synchronize Data

The user can load data with INSERT statements through JDBC protocol.

The use of INSERT statements is similar to that of INSERT statements in MySQL and other databases. INSERT statements support the following syntaxes:

```
 * INSERT INTO table SELECT ...
* INSERT INTO table VALUES(...)
```

Here we only introduce the second way. Refer to INSERT command file for more detailed instructions of INSERT command.

Write-once

Write-once means that the user directly executes an INSERT command. Examples are as follows:

```
INSERT INTO example_tbl (col1, col2, col3) VALUES (1000, "baidu", 3.25);
```

An INSERT command, for Palo, is a complete load transaction.

Therefore, whether a piece of data or multiple line data, we do not recommend loading data in the production environment with this method. The operation of high frequency word INSERT will lead to a large number of small files in the storage layer, which will seriously affect the system performance.

This method is only used for simple offline tests or low frequency operation.

Or the user can use the following methods to batch insert operations:

```
INSERT INTO example_tbl VALUES
(1000, "baidu1", 3.25)
(2000, "baidu2", 4.25)
(3000, "baidu3", 5.25);
```

We suggest that the number of inserts in a batch should be as large as possible, such as thousands or even 10,000 at a time. Or the user can use PreparedStatement to batch insert in the following way.

JDBC examples

Here is a simple example of JDBC batch INSERT codes:

```
package demo.palo;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class PaloJDBCDemo {

    private static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    private static final String DB_URL_PATTERN = "jdbc:mysql://%s:%d/%s?rewriteBatchedStatements=true";
    private static final String HOST = "127.0.0.1"; // Leader Node host
    private static final int PORT = 8030;   // http port of Leader Node
    private static final String DB = "example_db";
    private static final String TBL = "example_tbl";
    private static final String USER = "admin";
```

```
    private static final String USER = "admin";
    private static final String PASSWD = "my_pass";

    private static final int INSERT_BATCH_SIZE = 10000;

    public static void main(String[] args) {
        insert();
    }

    private static void insert() {
        // Don't ad semicolon ";" at the end
        String query = "insert into " + TBL + " values(?, ?)";
        // Set Label to be idempotent.
        // String query = "insert into " + TBL + " WITH LABEL my_label values(?, ?)";

        Connection conn = null;
        PreparedStatement stmt = null;
        String dbUrl = String.format(DB_URL_PATTERN, HOST, PORT, DB);
        try {
            Class.forName(JDBC_DRIVER);
            conn = DriverManager.getConnection(dbUrl, USER, PASSWD);
            stmt = conn.prepareStatement(query);

            for (int i =0; i < INSERT_BATCH_SIZE; i++) {
                stmt.setInt(1, i);
                stmt.setInt(2, i * 100);
                stmt.addBatch();
            }

            int[] res = stmt.executeBatch();
            System.out.println(res);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if (stmt != null) {
                    stmt.close();
                }
            } catch (SQLException se2) {
                se2.printStackTrace();
            }
            try {
                if (conn != null) conn.close();
            } catch (SQLException se) {
                se.printStackTrace();
            }
        }
    }
}
```

Please note:

1. The JDBC connection string needs to add the parameter `rewriteBatchedStatements=true` and use the method of `PreparedStatement`.

   Currently, Palo does not support PrepareStatemnt on the server side, so JDBC Driver will batch Prepare on the client side.

   `rewriteBatchedStatements=true` ensures Driver to perform batch precessing. Finally, the INSERT statements in the following form are sent to Palo:

```
INSERT INTO example_tbl VALUES
(1000, "baidu1", 3.25)
(2000, "baidu2", 4.25)
(3000, "baidu3", 5.25);
```

2. Batch size

   Pay attention that if a batch is too large, it will occupy the memory resources of the client because batch processing is carried out on the client.

   Palo will support PrepareStatemnt on the server in the future. Please wait and see.

3. Load atomicity

   INSERT operation itself, like other load methods, supports atomicity. Every INSERT operation is a load transaction, ensuring the write-in of all data atomicity in an INSERT.

   As mentioned earlier, we recommend that you load data in a "batch" way when loading data with INSERT, rather than single insert.

   Also, we can set a Label for each INSERT operation. Through Label mechanism, we can ensure the idempotence and atomicity of the operation, and finally achieve the data without loss and duplication. Refer to INSERT file for specific use of Label in INSERT.

Synchronize Data Through External Table

Palo can create external tables accessed through ODBC protocol. After the creation, the user can query the data of the external table directly through SELECT statement, or load the external table data through the method of `INSERT INTO SELECT`.

This document mainly introduces how to create external tables accessed through ODBC protocol and how to load these external table data. Currently supported data sources include:

- MySQL
- Oracle
- PostgreSQL

Create external table

Refer to CREATE ODBC TABLE syntax help manual for detailed introductions to creating ODBC external table.

Here, we describe the usage only by examples.

1. Create ODBC Resource

   ODBC Resource is to unify the management of connection information for external tables.

```
CREATE EXTERNAL RESOURCE `oracle_odbc`
PROPERTIES (
    "type" = "odbc_catalog",
    "host" = "192.168.0.1",
    "port" = "8086",
    "user" = "test",
    "password" = "test",
    "database" = "test",
    "odbc_type" = "oracle",
    "driver" = "Oracle"
);
```

We created here a Resource called `oracle_odbc`, whose type is `odbc_catalog`, indicating that this is a Resource used to store ODBC information. `odbc_type` is `oracle` , indicating this OBDC Resource is used to connect to Oracle database. Refer to Resource management document for other types of resources.

2. Create external table

```
CREATE EXTERNAL TABLE `ext_oracle_tbl` (
  `k1` decimal(9, 3) NOT NULL COMMENT "",
  `k2` char(10) NOT NULL COMMENT "",
  `k3` datetime NOT NULL COMMENT "",
  `k5` varchar(20) NOT NULL COMMENT "",
  `k6` double NOT NULL COMMENT ""
) ENGINE=ODBC
COMMENT "ODBC"
PROPERTIES (
    "odbc_catalog_resource" = "oracle_odbc",
    "database" = "test",
    "table" = "baseall"
);
```

We created here an external table called `ext_oracle_tbl`, quoting previously created `oracle_odbc` Resource.

Connect to Baidu cloud database RDS

1. Create RDS

   Create an RDS example through RDS quick start tutorial .

> Note: When creating RDS example, select the same network (VPC) as the Palo cluster at `Network type`->`Select network`. Availability zones can be different.

2. Create resources

```
 CREATE EXTERNAL RESOURCE `rds_odbc`
PROPERTIES (
    "type" = "odbc_catalog",
    "host" = "mysql56.rdsxxxxx.rds.gz.baidubce.com",
    "port" = "3306",
    "user" = "rdsroot",
    "password" = "12345",
    "odbc_type" = "mysql",
    "driver" = "MySQL"
);
```

The user need to alter parameters corresponding to `host` , `port` , `user` , `password` . host port can be checked in RDS example information. user and password should be acquired after creating accounts on RDS console.

3. Create external table

```
 CREATE EXTERNAL TABLE `mysql_table` (
  k1 int,
  k2 int
) ENGINE=ODBC
PROPERTIES (
    "odbc_catalog_resource" = "rds_odbc",
    "database" = "mysql_db",
    "table" = "mysql_tbl"
);
```

The user can carry out query and other operations after creating external tale.

Load data

1. Create Palo table

Here we create a Palo table with column information being the same as that of the external table `ext_oracle_tbl` created in the previous step:

```
CREATE EXTERNAL TABLE `palo_tbl` (
`k1` decimal(9, 3) NOT NULL COMMENT "",
`k2` char(10) NOT NULL COMMENT "",
`k3` datetime NOT NULL COMMENT "",
`k5` varchar(20) NOT NULL COMMENT "",
`k6` double NOT NULL COMMENT ""
)
COMMENT "Palo Table"
DISTRIBUTED BY HASH(k1) BUCKETS 2;
PROPERTIES (
   "replication_num" = "1"
);
```

Refer to CREATE-TABLE syntax help for detailed instructions of creating Palo table.

2. Load data (from table ext_oracle_tblto table palo_tbl)

```
INSERT INTO palo_tbl SELECT k1,k2,k3 FROM ext_oracle_tbl limit 100;
```

INSERT command is a synchronization command. Successful return indicates successful load.

Points of attention

- It is necessary to ensure that the external data source and Palo cluster are in the same VPC, and the Compute Node can be interworking with the network of the external data source.
- The essence of ODBC external tables is to access the data source through a single ODBC client, so it is improper to load a large amount of data at one time, loading in batches for several times is recommended.

Load JSON Data

Palo supports loading data in JSON format. This document mainly describes the precautions when loading JSON format data.

Supported load methods

Currently, only the following load methods support loading data in JSON format:

- load the local file in JSON format through STREAM LOAD.
- Subscribe and consume messages in JSON format in Kafka through ROUNTINE LOAD.

Other methods of data load in JSON format are not supported for the time being.

Supported Json format

Currently, only the following two Json formats are supported:

1. Multi-row data represented by Array

   Json format with Array as root node. Each element in Array represents a row of data to be loaded, which is usually an Object. Examples are as follows:

```
[
  { "id": 123, "city" : "beijing"},
  { "id": 456, "city" : "shanghai"},
  ...
]
```

```
[
  { "id": 123, "city" : { "name" : "beijing", "region" : "haidian"}},
  { "id": 456, "city" : { "name" : "beijing", "region" : "chaoyang"}},
  ...
]
```

This method is usually used in Stream Load mode to represent multiple rows of data in a batch of loaded data.

The use of this method must be accompanied with the setting of `stripe_outer_array=true`. During parsing, Palo will expand the array and then parse each Object as a row of data in turn.

2. Single row data represented by Object

   Json format with Object as root node. The entire Object represents a row of data to be loaded. Examples are as follows:

```
{ "id": 123, "city" : "beijing"}
```

```
{ "id": 123, "city" : { "name" : "beijing", "region" : "haidian" }}
```

This method is usually used for Routine Load method, for example, representing a message in Kafka, i.e., a row of data.

fuzzy_parse parameter

In STREAM LOAD , the user can add `fuzzy_parse` parameter to accelerate load efficiency of JSON data.

This parameter is usually used to load **multi-row data represented by Array** as the format, so it is usually combined with `strip_outer_array=true` to use.

This function requires that in each data row of Array, **the field order be completely consistent**. Palo will only parse according to the field order in the first row, and then access the subsequent data in the form of subscripts. This method can improve the load efficiency by 3-5 times.

Json Path

Palo supports extracting specified data in Json through Json path.

Note: for Array type data, Palo will first expand the array, and finally perform single row processing according to Object format. Therefore, the examples after this document are illustrated with Json data in single Object format.

- Json Path is not specified

  If there is no specified Json path, Palo will, by default, search the elements in Object with the column names in the table. Examples are as follows:

  The table contains two columns: `id`, `city`

  Json data are as follows:

```
{ "id": 123, "city" : "beijing"}
```

Then Palo will match with `id`, `city` to get the final data `123` and `beijing`.

If Json data are as follows:

```
{"id": 123, "name" : "beijing"}
```

Then Palo will match with `id`, `city` to get the final data `123` and `null`.

- Json Path is specified

  Specify a set of Json Paths in the form of Json data, Each element in the array represents a column to be extracted. Examples are as follows:

```
["$.id", "$.name"]
```

```
["$.id.sub_id", "$.name[0]", "$.city[0]"]
```

  Palo will use the specified Json Path for data matching and extraction.

- Match non-basic types

  The final matching values in the previous examples are all basic types, such as integer, string, etc. Palo currently does not support composite types like Array, Map, etc. So when matching a non-basic type, Palo will convert the type to a string in Json format and load it as a string type. Examples are as follows:

  Json data are:

```
{ "id": 123, "city" : { "name" : "beijing", "region" : "haidian" }}
```

Json Path is ["$.city"], then the matched elements are:

```
{ "name" : "beijing", "region" : "haidian" }
```

The element will be converted to a string for subsequent load:

```
"{'name':'beijing','region':'haidian'}"
```

- Match failed

  When the match fails, the value will return tonull. Examples are as follows:

  Json data are:

```
{ "id": 123, "name" : "beijing"}
```

  Json Path is ["$.id", "$.info"]. then the matched elements are: 123 and null.

  Palo currently does not distinguish between null values expressed in Json data and the ones generated from failed match.

  Suppose that Jason data are:

```
{ "id": 123, "name" : null }
```

then the same results `123` and `null`will be acquired when using the following two types of Json Path.

```
["$.id", "$.name"]
```

```
["$.id", "$.info"]
```

- Complete failed match

  In order to prevent misoperation caused by some parameter setting errors, when Palo tries to match a row of data and if all the columns fail to match, the row will be considered as an error row. Suppose that Json data are:

```
{ "id": 123, "city" : "beijing" }
```

If Json Path is incorrectly written as （or if Json Path is not specified, the columns in the table do not contain `id` and `city`）：

```
["$.ad", "$.info"]
```

then it will lead to a complete failed match, and the row will be marked as an error row instead of producing `null, null`.

Json Path and Columns

Json Path is used to specify how to extract data in Json format, while Columns specifies the mapping and converting relationship of columns. The two can be used in combination.

In other words, it is equivalent to rearrange the columns of data in Json format according to the column order specified in Json Path through Json Path. Afterwards, mapping can be achieved for the rearranged source data and the columns of the table through Columns. Examples are as follows:

Data contents:

```
{"k1" : 1, "k2": 2}
```

Table structure:

`k2 int, k1 int`

load statement 1 (Take Stream Load as an example) :

```
curl -v --location-trusted -u root: -H "format: json" -H "jsonpaths: [\"$.k2\", \"$.k1\"]" -T example.json
http://127.0.0.1:8030/api/db1/tbl1/_stream_load
```

In the load statement 1, only Json Path is specified while Columns is not. And the function of Json Path is to extract the Json data according to the field order in Json Path, and then write them according to the order of the table structure. The final result of the loaded data is as follows:

```
 +------+------+
| k1   | k2   |
+------+------+
|    2 |    1 |
+------+------+
```

It can be seen that actually k1 column is loaded with the "k2" column value in Json data. The reason for this is that the filed name in Json is different from that in the table structure. We need to explicitly specify the mapping relationship between the two.

load statement 2:

```
curl -v --location-trusted -u root: -H "format: json" -H "jsonpaths: [\"$.k2\", \"$.k1\"]" -H "columns: k2, k1" -T example.json http://127.0.0.1:8030/api/db1/tbl1/_stream_load
```

Compared with the load statement 1, the Columns field is added here to show the mapping relationship of columns in the order of `k2, k1`, which means that after extracting data in the order of fields in Json Path , the value of the first column is specified as that of column k2 in the table, and the value of the second column is specified as that of column k1 in the table. The final result of the load data is as follows:

```
 +------+------+
| k1   | k2   |
+------+------+
|    1 |    2 |
+------+------+
```

Of course, converting column in Columns, like other loads, can be performed. Examples are as follows:

```
curl -v --location-trusted -u root: -H "format: json" -H "jsonpaths: [\"$.k2\", \"$.k1\"]" -H "columns: k2, tmp_k1, k1 = tmp_k1 * 100" -T example.json http://127.0.0.1:8030/api/db1/tbl1/_stream_load
```

The above example will load k1 value after multiplying it by 100. The final result of the load data is as follows:

```
 +------+------+
| k1   | k2   |
+------+------+
|  100 |    2 |
+------+------+
```

The values of NULL and Default

The example data are as follows:

```
[
  {"k1": 1, "k2": "a"},
  {"k1": 2},
  {"k1": 3, "k2": "c"},
]
```

The table structure is: `k1 int null, k2 varchar(32) null default "x"`

load statement is as follows:

```
curl -v --location-trusted -u root: -H "format: json" -H "strip_outer_array: true" -T example.json
http://127.0.0.1:8030/api/db1/tbl1/_stream_load
```

The user may expect the following load results, for the missing columns, fill in the default values.

```
 +------+------+
| k1   | k2   |
+------+------+
|    1 |    a |
+------+------+
|    2 |    x |
+------+------+
|    3 |    c |
+------+------+
```

The actual load result, however, is as follows, which means NULL is added to the missing column.

```
 +------+------+
| k1   | k2   |
+------+------+
|    1 |    a |
+------+------+
|    2 | NULL |
+------+------+
|    3 |    c |
+------+------+
```

The reason for this is that Palo, through the information of load statement, does not know that "the missing column is the k2 column in the table" . If the user wants to load the above data according to the expected result, the load statement should be as follows:

```
curl -v --location-trusted -u root: -H "format: json" -H "strip_outer_array: true" -H "jsonpaths: [\"$.k1\", \"$.k2\"]" -H "columns: k1, tmp_k2, k2 = ifnull(tmp_k2, 'x')" -T example.json http://127.0.0.1:8030/api/db1/tbl1/_stream_load
```

Application examples

Stream Load

Due to the non-splitting property of the Json format, when loading files in the Json format with Stream Load, processing will start after the whole file contents is loaded into the memory. Therefore, if the file is too large, it may take up more memory.

Suppose the table structure is:

```
 id      INT     NOT NULL,
city    VARHCAR NULL,
code    INT     NULL
```

1. load single row data 1

```
{"id": 100, "city": "beijing", "code" : 1}
```

- Json Path is not specified

```
curl --location-trusted -u user:passwd -H "format: json" -T data.json
http://localhost:8030/api/db1/tbl1/_stream_load
```

load result:

```
100    beijing    1
```

- Json Path is specified

```
curl --location-trusted -u user:passwd -H "format: json" -H "jsonpaths: [\"$.id\",\"$.city\",\"$.code\"]" -T
data.json http://localhost:8030/api/db1/tbl1/_stream_load
```

load result:

```
100    beijing    1
```

2. load single row data 2

```
{"id": 100, "content": {"city": "beijing", "code" : 1}}
```

- Json Path is specified

```
curl --location-trusted -u user:passwd -H "format: json" -H "jsonpaths:
[\"$.id\",\"$.content.city\",\"$.content.code\"]" -T data.json http://localhost:8030/api/db1/tbl1/_stream_load
```

load result:

```
100    beijing    1
```

3. load multi row data

```
[
  {"id": 100, "city": "beijing", "code" : 1},
  {"id": 101, "city": "shanghai"},
  {"id": 102, "city": "tianjin", "code" : 3},
  {"id": 103, "city": "chongqing", "code" : 4},
  {"id": 104, "city": ["zhejiang", "guangzhou"], "code" : 5},
  {
    "id": 105,
    "city": {
      "order1": ["guangzhou"]
    },
    "code" : 6
  }
]
```

- Json Path is specified

```
curl --location-trusted -u user:passwd -H "format: json" -H "jsonpaths: [\"$.id\",\"$.city\",\"$.code\"]" -H
"strip_outer_array: true" -T data.json http://localhost:8030/api/db1/tbl1/_stream_load
```

load result:

```
 100    beijing                 1
101    shanghai                 NULL
102    tianjin                 3
103    chongqing                4
104    ["zhejiang","guangzhou"]    5
105    {"order1":["guangzhou"]}    6
```

4. Convert loaded data

   The data are still the multi row data in example 3, now the user needs to add 1 to `code` column in the load data to load.

```
curl --location-trusted -u user:passwd -H "format: json" -H "jsonpaths: [\"$.id\",\"$.city\",\"$.code\"]" -H
"strip_outer_array: true" -H "columns: id, city, tmpc, code=tmpc+1" -T data.json
http://localhost:8030/api/db1/tbl1/_stream_load
```

   load result:

```
 100    beijing                 2
101    shanghai                 NULL
102    tianjin                 4
103    chongqing                5
104    ["zhejiang","guangzhou"]    6
105    {"order1":["guangzhou"]}    7
```

Routine Load

The principle of processing Json data by Route Load is the same as that by Stream Load, which will not be repeated here.

For Kafka data sources, the content in each Message is regarded as a complete Json data. If there are multiple rows data in Array format in a Massage, multiple rows will be loaded, and the offset of Kafka will only be added by 1. While if an Array format Json represents multiple rows data, the error row will be only added by 1 if parsing of Json fails because of the wrong Json format (Palo can't actually judge how many rows of data it contains because of the parsing failure, it can only record one row of error data).

Load Transaction and Atomicity

Load atomicity

All Load operations in Palo have atomicity guarantee, that is, the data in an load job are either all succeed or all fail instead of partial imported.

We can also implement atomic Load of multi table in BROKER LOAD .

For attached materialized view, the atomicity and consistency with the base table are also guaranteed.

Label mechanism

The load job of Palo can be set with a label, which is usually a user-defined string with certain business logic attributes.

The main function of Label is to uniquely identify an Load task and ensure that the same Label will be successfully imported

only once.

Label mechanism can ensure that the Load data are not lost or duplicated. If the upstream data source can guarantee At-Least-Once semantic, then it can guarantee Exactly-Once semantic with the Label mechanism of Palo.

Label is unique in a database. Its default retention period is 3 days, which means that the completed Label will be automatically cleared after 3 days and the Label can be reused .

The best practices

Usually, Label is set to the format of `business logic+time` like `my_business1_20201010_125000`.

This Label usually represents a batch of data generated from business `my_business1` at `2020-10-10 12:50:00`. Through this Label setting, Load task status of business can be queried through Label to know clearly whether the data of the batch at this time point has been loaded successfully. Try to Load again with this Label if fails.

Column mapping, converting and filtering

Palo supports rich column mapping, converting, and filtering operations to flexibly deal with the original data to be loaded.

This document mainly describes how to load data with these functions.

General introductions

The data processing steps of Palo in loading are as follows:

1. The data are read into Palo in column order in the original file.
2. The original data are filtered for the first time through PRECEDING FILTER condition.
3. Through column mapping and converting, the original data are mapped to the target column order.
4. The converted data are filtered again by post filter condition (WHERE).
5. Write final data.

Column mapping, converting and filtering parameters are optional in the load job. In default vacancy, Palo will split the rows in the source file according to the default column separator `\t`, and then correspond them to the table columns in order. If the number of columns in the source file does not match the number of columns in the table, data quality problems will occur and the data cannot be loaded. In this case, we need to explicitly describe the column mapping, converting and filtering information.

Supported load methods

- BROKER LOAD

```
 LOAD LABEL example_db.label1
(
    DATA INFILE("bos://bucket/input/file")
    INTO TABLE `my_table`
    (k1, k2, tmpk3)
    PRECEDING FILTER k1 = 1
    SET (
        k3 = tmpk3 + 1
    )
    WHERE k1 > k2
)
WITH BROKER bos
(
    ...
);
```

- STREAM LOAD

```
 curl
--location-trusted
-u user:passwd
-H "columns: k1, k2, tmpk3, k3 = tmpk3 + 1"
-H "where: k1 > k2"
-T file.txt
http://host:port/api/testDb/testTbl/_stream_load
```

- ROUTINE LOAD

```
 CREATE ROUTINE LOAD example_db.label1 ON my_table
COLUMNS(k1, k2, tmpk3, k3 = tmpk3 + 1),
PRECEDING FILTER k1 = 1,
WHERE k1 > k2
...
```

All the above load methods support column mapping, converting and filtering of source data:

- Preceding filter: Filter the original data that have been read for the first time.

```
 PRECEDING FILTER k1 = 1
```

- Mapping: Define the columns in source data. If the defined column name is the same as the column in the table, it will be mapped directly to the column in the table. If they are different, the defined column can be used for subsequent converting. As is shown in the example above:

```
(k1, k2, tmpk3)
```

- Converting: Convert the mapped columns in the first step with built-in expressions, functions, and self-defined functions, and remap them to the corresponding columns in the table. As is shown in the example above:

```
k3 = tmpk3 + 1
```

- Post filter: Filter the mapped and converted columns with expressions. Filtered data rows will not be loaded into the system. As is shown in the example above:

```
WHERE k1 > k2
```

Column mapping

Column mapping mainly aims to describe the information of each column in the loaded file, which is equivalent to defining the column name in the source data. We can load the source files with different order and number of columns in the table into Palo through the description of column mapping relationship. Here are the examples:

Suppose there are 4 columns in the source file, as is shown below (The column name of the header is only for convenience, actually there is no header):

Column 1	Column 2	Column 3	Column 4
1	100	beijing	1.1
2	200	shanghai	1.2
3	300	guangzhou	1.3
4	\N	chongqing	1.4

> Note: `\N` means null in source file.

1. Adjust mapping order

   Suppose there are 4 columns `k1,k2,k3,k4` in the table. The expected load mapping relationship is as follows:

```
 Column 1 -> k1
Column 2 -> k3
Column 3 -> k2
Column 4 -> k4
```

   Then the writing order of column mapping should be as follows:

```
(k1, k3, k2, k4)
```

2. There are more columns in the source file than in the table

   Suppose there are 3 columns `k1,k2,k3` in the table. The expected load mapping relationship is as follows:

```
 Column 1 -> k1
Column 2 -> k3
Column 3 -> k2
```

Then the writing order of column mapping should be as follows:

```
(k1, k3, k2, tmpk4)
```

Where `tmpk4` is a self-defined and nonexistent column name which can be ignored by Palo in the table.

3. If there are less columns in the source file than in the table, fill in with default values.

   Suppose there are 5 columns `k1,k2,k3,k4,k5` in the table. The expected load mapping relationship is as follows:

```
 Column 1 -> k1
Column 2 -> k3
Column 3 -> k2
```

   Here we only use the first 3 columns in the source file. Columns `k4,k5` are expected to be filled in with default values.

   Then the writing order of column mapping should be as follows:

```
(k1, k3, k2)
```

   If columns `k4,k5` have default values, the default values will be filled in. Otherwise, if it is a `nullable` column, then `null` value will be filled in. Otherwise, the load job will report an error.

Column preceding filter

Preceding filter aims to filter, for the first time, the original data that have been read. At present, only BROKER LOAD and ROUTINE LOAD are supported.

Preceding filter is applied in the following scenarios:

1. Filter before converting

   Scenarios that are expected to be filtered before column mapping and converting. Some unnecessary data can be filtered out first.

2. Filter column does not exist in table and is only used as filter ID

   The data of multiple tables, for example, are stored in the source data (Or the data of multiple tables are written to the same Kafka message queue). Each row in the data has a list name to identify which table the data in this row belongs to. Users can screen the corresponding table data to load them through preceding filtering conditions.

Column converting

The column converting allows users to convert the column values in the source file. Currently, Palo supports most built-in functions and self-defined functions for converting.

> Note: The self-defined function belongs to a certain database, so the user needs read permission to the database to convert when using self-defined functions.

Converting are usually defined with column mapping, which means to map the columns first and then convert them. Here are the examples:

Suppose there are 4 columns in the source file, as is shown below (The column name of the header is only for convenience, actually there is no header):

Column 1	Column 2	Column 3	Column 4
1	100	beijing	1.1
2	200	shanghai	1.2
3	300	guangzhou	1.3
4	400	chongqing	1.4

1. Load column values from source file into table after being converted

   Suppose there are 4 columns `k1,k2,k3,k4` in the table. The expected load mapping and converting relationships are as follows:

   ```
    Column 1       -> k1
   Column 2 * 100 -> k3
   Column 3       -> k2
   Column 4       -> k4
   ```

   Then the writing order of column mapping should be as follows:

   ```
   (k1, tmpk3, k2, k4, k3 = tmpk3 * 100)
   ```

   This is equivalent to naming the second column in the source file as `tmpk3`, and specifying the value of column `k3` in the table as `tmpk3 * 100`. The final data in the table are as follows:

k1	k2	k3	k4
1	beijing	10000	1.1
2	shanghai	20000	1.2
3	guangzhou	30000	1.3
null	chongqing	40000	1.4

2. Conditional column converting can be achieved through case when function.

   Suppose there are 4 columns `k1,k2,k3,k4` in the table. We hope to load `beijing, shanghai, guangzhou, chongqing` in the source data after respectively being converted to the corresponding region ID:

   ```
    Column 1                 -> k1
   Column 2                 -> k2
   Column 3 after being converted to region id    -> k3
   Column 4                 -> k4
   ```

   Then the writing order of column mapping should be as follows:

   ```
   (k1, k2, tmpk3, k4, k3 = case tmpk3 when "beijing" then 1 when "shanghai" then 2 when "guangzhou" then 3 when "chongqing" then 4 else null end)
   ```

   The final data in the table are as follows:

k1	k2	k3	k4
1	100	1	1.1
2	200	2	1.2
3	300	3	1.3
null	400	4	1.4

3. Convert the null value in the source file to 0 and then load. And convert region id in example 2.

Suppose there are 4 columns `k1,k2,k3,k4` in the table. We wish to convert the null value in column K1 to 0 and then load when converting region id:

```
 Column 1 If null, convert to 0   -> k1
Column 2                  -> k2
Column 3                  -> k3
Column 4                  -> k4
```

Then the writing order of column mapping should be as follows:

```
 (tmpk1, k2, tmpk3, k4, k1 = ifnull(tmpk1, 0), k3 = case tmpk3 when "beijing" then 1 when "shanghai" then 2 when "guangzhou" then 3 when "chongqing" then 4 else null end)
```

The final data in the table are as follows:

k1	k2	k3	k4
1	100	1	1.1
2	200	2	1.2
3	300	3	1.3
0	400	4	1.4

Column filter

We can, after column mapping and converting, filter the data that we don't want to load into Palo through filtering conditions. Examples are as follows:

Suppose there are 4 columns in the source file, as is shown below (The column name of the header is only for convenience, actually there is no header):

Column 1	Column 2	Column 3	Column 4
1	100	beijing	1.1
2	200	shanghai	1.2
3	300	guangzhou	1.3
4	400	chongqing	1.4

1. Filter directly in the case of column mapping and converting default

   Suppose there are 4 columns `k1,k2,k3,k4` in the table. We can directly define filter conditions in the case of column mapping and converting default. If we want to load only the data rows with column 4 larger than 1.2 in the source file, the filter conditions are as follows:

   ```
   where k4 > 1.2
   ```

   The final data in the table are as follows:

k1	k2	k3	k4
3	300	guangzhou	1.3
null	400	chongqing	1.4

   By default, Palo maps columns in order, so the fourth column in the source file is automatically mapped to column `k4` in the table.

2. Filter the data after column converting

   Suppose there are 4 columns `k1,k2,k3,k4` in the table. We converted the province names to id in **Column converting**

example. Here we want to filter out the data with id being 3. Converting and filter conditions are as follows:

```
 (k1, k2, tmpk3, k4, k3 = case tmpk3 when "beijing" then 1 when "shanghai" then 2 when "guangzhou" then 3 when "chongqing" then 4 else null end)
where k3 != 3
```

The final data in the table are as follows:

k1	k2	k3	k4
1	100	1	1.1
2	200	2	1.2
null	400	4	1.4

Here we can see that the column values when filtering are the final column values after mapping and converting, not the original data.

3. Multi-condition filtering

   Suppose there are 4 columns `k1,k2,k3,k4` in the table. If we want to filter out the data whose column `k1` is `null` and the data whose column `k4` is less than 1.2, the filter conditions are as follows:

   ```
   where k1 is null and k4 < 1.2
   ```

   The final data in the table are as follows:

k1	k2	k3	k4
2	200	2	1.2
3	300	3	1.3

Data quality issues and filter thresholds

There are three types of data rows to be processed in the load job:

1. Filtered Rows

   Data filtered out due to unqualified data quality. Unqualified data quality includes data format problems such as type error, precision error, super long string length, mismatched number of file columns, and data rows filtered out due to no corresponding sections.

2. Unselected Rows

   Data rows filtered out due to filter conditions of columns `preceding filter` or `where`.

3. Loaded Rows

   Data rows that are correctly loaded.

Palo load task allows users to set the maximum error rate (`max_filter_ratio`). If error rate of loaded data is lower than the threshold, these error rows will be ignored and other correct data will be loaded.

How to calculate error rate:

```
##### Filtered Rows / (#Filtered Rows + #Loaded Rows)
```

In other words, `Unselected Rows` will not participate in calculating error rate.

Strict Mode

Strict mode (strict_mode) is a parameter configuration in load operation. This parameter will affect the load behavior of some

values and the final loaded data.

This document mainly describes how to set strict mode and the impact of strict mode.

How to set

Default setting of strict mode is False, which means OFF status.

Different load methods have different ways to set strict mode.

1. BROKER LOAD

```
LOAD LABEL example_db.label1
(
    DATA INFILE("bos://my_bucket/input/file.txt")
    INTO TABLE `my_table`
    COLUMNS TERMINATED BY ","
)
WITH BROKER bos
(
    "bos_endpoint" = "http://bj.bcebos.com",
    "bos_accesskey" = "xxxxxxxxxxxxxxxxxxxxxxxxx",
    "bos_secret_accesskey"="yyyyyyyyyyyyyyyyyyyyyyyy"
)
PROPERTIES
(
    "strict_mode" = "true"
)
```

2. STREAM LOAD

```
curl --location-trusted -u user:passwd \
-H "strict_mode: true" \
-T 1.txt \
http://host:port/api/example_db/my_table/_stream_load
```

3. ROUTINE LOAD

```
CREATE ROUTINE LOAD example_db.test_job ON my_table
PROPERTIES
(
    "strict_mode" = "true"
)
FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic"
);
```

4. INSERT

   Set through Session variables:

```
SET enable_insert_strict = true;
INSERT INTO my_table ...;
```

Functions of strict mode

Strict mode means to carry out strict filtering for column type converting during load.

The strategy of strict filtering is as follows:

For column type converting, the error data will be filtered if strict mode is started. The error data here refer to the kind of data whose original data are not `null` but whose result are `null` after column type converting.

"`Column type converting`" here does not include `null` value calculated by functions.

If the loaded column type contains range restriction and if the original data can pass the type converting normally but cannot pass the range restriction, the strict mode will not affect it. For example, if the type is `decimal(1,0)`, and the original data is 10, then it belongs to the range that can pass the type converting but not to the column statement. This data can not be affect by strict mode.

1. Take column type Tinyint as an example:

Original data type	original data example	Value converted to Tinyint	Strict mode	Results
Null	\N	NULL	On or off	NULL
Non-null	"abc" or 2000	NULL	On	Invalid value（filtered）
Non-null	"abc"	NULL	Off	NULL
Non-null	1	1	On or off	Correct load

> Notes:
>
> 1. Null values are allowed to be loaded in table columns.
> 2. After `abc` and `2000` are converted to TinyInt, they will become Null due to type or precision. These data will be filtered when strict mode is on. And `null` will be loaded if strict mode is off.

2. Take column type Decimal(1,0) as an example

Original data type	original data example	Value converted to Decimal	Strict mode	Results
Null	\N	null	On or off	NULL
Non-null	aaa	NULL	On	Invalid value（filtered）
Non-null	aaa	NULL	Off	NULL
Non-null	1 or 10	1 or 10	On or off	Correct load

> Notes:
>
> 1. Null values are allowed to be loaded in table columns.
> 2. After `abc` is converted to Decimal, it will become Null due to the type. These data will be filtered when strict mode is on. And `null` will be loaded if strict mode is off.
> 3. Although `10` is a value beyond the range, strict mode has no effect on it because its type meets the requirements of decimal, and `10` will be finally filtered in other load processes, but it will not filtered by strict mode.

Data Update and Delete

Data Update

The data stored in Palo enter the system in the form of Append, which means that all the written data are unchangeable.

Thus, Palo uses **mark** to update the data. That is, in a batch of updated data, mark the previous data as **Delete** and write new data.

In reading process, Palo will automatically process these marked data (Merge-on-Read) to ensure that the user reads the latest data. At the same time, the data merging (Compaction) thread in Palo background will continue to merge the data and eliminate **marked** data to reduce the merging operation in the reading process and speed up the query.

Most scenarios to alter data are only applicable to Unique Key data model because only this model can guarantee the uniqueness of the primary key to support data update according to the primary key.

This document mainly introduces how to update data with Unique Key data model.

Data update

Please refer to the relevant documents for the description of UNIQUE KEY and we will not repeat here. The following is just an example.

1. Create a table of UNIQUE KEY model

```
CREATE TABLE order_table
(
    order_id BIGINT,
    order_type VARCHAR(8),
    order_status VARCHAR(32)
)
UNIQUE KEY(order_id)
DISTRIBUTED BY HASH(order_id) BUCKETS 8;
```

2. Import the first batch of data

```
1000, TYPE#1, PAID
1001, TYPE#2, PENDING
1002, TYPE#3, PAID
```

3. Alter field `order_status` of the order entry with ID 1001 to `PAID`, and the following data should be loaded:

```
1001, TYPE#2, PAID
```

Palo will alter the field `order_status` of this entry to PAID in the process of reading or background merging according to primary key 1001.

Update partial fields

In the previous example, we just need to update `order_status` field, but the loaded data need to contain `order_type` field.

In some scenarios, users only know the primary key and the values of partial fields to be updated rather than the whole column data. In this case, we can obtain the whole column data through aggregation method `REPLACE_IF_NOT_NULL`.

UNIQUE_ KEY model, in essence, is a kind of aggregation model, and its aggregation type on non-primary key columns is `REPLACE` by default. For `REPLACE_IF_NOT_NULL`, it means no update when the value is `null`. Examples are as follows:

1. Create a table of UNIQUE KEY model with the method of `REPLACE_IF_NOT_NULL`.

```
CREATE TABLE order_table
(
    order_id BIGINT,
    order_type VARCHAR(8),
    order_status VARCHAR(32)
)
UNIQUE KEY(order_id)
DISTRIBUTED BY HASH(order_id) BUCKETS 8
PROPERTIES
(
    "replace_if_not_null" = "true"
);
```

2. Import the first batch of data

```
1000, TYPE#1, PAID
1001, TYPE#2, PENDING
1002, TYPE#3, PAID
```

3. Alter field `order_status` of the order entry with ID 1001 to `PAID`, and the following data should be loaded:

```
1001, \N, PAID
```

`\N` in original data means null. Palo will alter the field `order_status` of this entry to PAID in the process of reading or background merging according to primary key 1001. And the field `order_type` will not be replaced because it is null.

Update sequence

Within Palo, only two batches of loaded data can be guaranteed, and the data of the later batch will cover and update the data of the previous batch. However, if multiple line records with the same primary key appear in the same batch of data, Palo cannot identify which one is the final effective data.

Suppose that the loaded data of a certain batch are as follows:

```
1000, TYPE#1, PENDING
1001, TYPE#2, PENDING
1000, TYPE#3, PAID
```

Note that the primary keys of the first line and the third line are the same, so it is impossible to determine which one will take effect, and the status of the order 1000 queried by the user may be `PENDING` or `PAID`.

The business side, to solve the problem, needs to ensure that there are no lines with the same primary key in the same batch of data. Or refer to Sequence Column to carry out data adaptation.

Data Delete

Palo delete data in the following ways:

- TRUNCATE, this command aims to clear the table or section directly, but not to delete the corresponding metadata. It is

suggested to use this command when the operation cost is low and when there is a need to clear the data.

- DROP, this command aims to delete table or section, together with data and metadata simultaneously.
- DELETE, Delete statement aims to delete data according to conditions. Refer to the section**Delete according to conditions** in this document for details.
- MARK DELETE, Mark Delete conducts the function of delete data by line. Refer to the section**Delete mark** in this document for details.

This document mainly introduces two methods of DELETE and BATCH DELETE. Please refer to the corresponding command documents for other methods.

Delete according to conditions

Delete data according to conditions with `DELETE` command. Refer to DELETE command file for detailed instructions. Examples are as follows:

```
DELETE FROM example_table WHERE event_day < 20201001 AND event_key != 1000;

DELETE FROM example_table PARTITION p202010 WHERE event_key in (1000, 1001, 1002, 1003);
```

DELETE command is a synchronous command, and a successful return means a successful deletion.

View the historical DELETE operation records through the following command:

```
mysql> SHOW DELETE FROM example_db;
+-----------+---------------+---------------------+-----------------+----------+
| TableName | PartitionName | CreateTime          | DeleteCondition | State    |
+-----------+---------------+---------------------+-----------------+----------+
| empty_tbl | p3            | 2020-04-15 23:09:35 | k1 EQ "1"       | FINISHED |
| test_tbl  | p4            | 2020-04-15 23:09:53 | k1 GT "80"      | FINISHED |
+-----------+---------------+---------------------+-----------------+----------+
2 rows in set (0.00 sec)
```

Refer to SHOW DELETE command file for specific instructions.

Points for attention

- DELETE command is not suitable for high-frequency deletion operations, sending a large number of DELETE commands in a

short time, for example, will seriously affect the underlying data merging and query efficiency. DELETE operation, in essence, is to store a deletion condition, which will be applied to filter every row of records during query, therefore, the query efficiency will be reduced when there are a large number of deletion conditions.

- Try to avoid using the alternate execution mode of `DELETE-LOAD-DELETE-LOAD`. This mode is rather unfriendly to the underlying data merging strategy, which may give rise to a large amount of unmerged data and cause backlog.

Mark delete

Mark delete function is mainly used in scenarios that need real-time update and synchronization like Synchronize MySQL Binlog data. This method can only be applied to the tables of UNIQUE KEY model. Refer to MARK DELETE document for details.

Mark Deletion

Mark deletion function is a supplement to the deletion function of DELETE statement. Deleting data with DELETE statement cannot support high-frequency operation scenarios.

Additionally, similar to the CDC (Change Data Capture) scenario, INSERT and DELETE usually appear alternately.

Mark deletion function is designed to support the above two scenarios.

Implementation principle

Mark deletion function only applies to tables of UNIQUE KEY model. The principle is to add a hidden column `__DELETE_mark__` to the table. If the column value is true, it means that the row is carried out a deletion operation, if it is false, it means that the row is carried out an insert operation. The user only needs to add in the loaded data to hide the column.

Suppose the table structure is as follows:

Column name	Primary key
order_id	Yes
order_type	No
order_status	No

Insert the following three data lines:

```
 1000, TYPE#1, PENDING, false
1001, TYPE#2, PENDING, false
1002, TYPE#3, PENDING, false
```

The values of the fourth columns are all false, which means that the three lines are all carried out insert operations. The query results after execution are as follows:

order_id	order_type	order_status
1000	TYPE#1	PENDING
1001	TYPE#2	PENDING
1002	TYPE#3	PENDING

Next, load two more lines of data:

```
 1001, TYPE#2, PENDING, true
1002, TYPE#3, PAID, false
```

The first line is carried out a deletion operation, which is deleting the data corresponding to the order ID 1001（The 2 values TYPE#2, PENDING are meaningless in this data line and can be filled in at will, but they must be filled in to ensure that the number of columns matches）.

The second line is carried out an insert operation, which is equivalent to updating the status of the order 1002 to `PAID`.

The query results after execution are as follows:

order_id	order_type	order_status
1000	TYPE#1	PENDING
1002	TYPE#3	PAID

As seen from the above examples that the user can realize the update or deletion operation of primary key in conjunction with the UNIQUE KEY model. Set the corresponding hidden column as true to delete, while the value of hidden column is false, the operation is similar to UPSERT operation, that is, **update if there is any, insert if there is none**.

Enable Mark deletion function

Mark deletion is a new function introduced after Palo version 3.10.

The UNIQUE KEY tables created in the new version contain hidden columns by default. While the UNIQUE KEY tables created in previous versions do not have hidden columns. For tables of old versions, add hidden columns in the following ways:

```
ALTER TABLE tablename ENABLE FEATURE "BATCH_DELETE";
```

This operation, in essence, is a Schema Change operation , refer to SHOW ALTER TABLE COLUMN to check job execution progress after execution.

Set a variable to show hidden columns if you want to determine whether a table has enabled Mark deletion function.

```
SET show_hidden_columns=true`
```

Then use `DESC tablename`, if there is a column `__DELETE_mark__` in output, it means that the table has enabled Mark deletion function.

Use mark deletion function in load

The methods of using Mark deletion function in different data load methods are slightly different. Mark deletion currently

supports the following data load methods:

- STREAM LOAD
- BROKER LOAD
- ROUTINE LOAD

Add the following two attributes to use Mark deletion function in these loads:

1. Merge Type

   Merge Type has 3 types, APPEND, DELETE and MERGE, and APPEND is the default.

   APPEND method has no difference from normal load.

   DELETE method means that each row in the batch of data is carried out a deletion operation. And there is no need to specify the column Delete Label in this method, .

   MERGE method means that this batch of data is carried out the mixed operations of both insert and deletion. At this time, the user needs to identify the operation type of each row by specifying the column Delete Label.

2. Delete Label

   Delete Label aims to, in MERGE method, specify that a certain column in the data is the column with deletion label. The user can specify the value of this column as DELETE operation.

Refer to the respective documents for specific usage syntax. Here, we only give simple examples of different load modes. Suppose that the original loaded data are as follows:

```
 1000,TYPE#1,PENDING,0
1001,TYPE#2,PENDING,0
1002,TYPE#3,PENDING,0
1003,TYPE#2,PENDING,1
1004,TYPE#3,PAID,1
```

Stream Load

The request of Stream Load is as follows :

```
 curl --location-trusted -u root \
-H "columns: order_id, order_type, order_status, delete_label" \
-H "merge_type: MERGE" \
-H "delete: delete_label=1" \
-T data.txt http://host:port/api/testDb/testTbl/_stream_load
```

This example shows that the fourth column is Delete Label column, and the corresponding line is carried out deletion operation when the value is 1.

Broker Load

```
LOAD LABEL example_db.my_label
(
    MERGE DATA INFILE("hdfs://abc.com:8888/user/palo/test/ml/file1")
    INTO TABLE example_tbl
    COLUMNS TERMINATED BY ","
    (order_id, order_type, order_status, delete_label)
    DELETE ON delete_label=1
)
WITH BROKER 'bos'
(
    ...
);
```

This example shows that the fourth column is Delete Label column, and the corresponding line is carried out deletion operation when the value is 1.

Routine Load

```
CREATE ROUTINE LOAD example_db.job1 ON example_tbl
WITH DELETE
COLUMNS(order_id, order_type, order_status, delete_label)
PROPERTIES
(
    "desired_concurrent_number"="3",
    "max_batch_interval" = "20",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200",
    "strict_mode" = "false"
)
FROM KAFKA
(
    ...
);
```

Note that the Merge Type we use here is DELETE, so we don't need to specify Delete Label, the value of the fourth column will not be used, and all data are carried out deletion operation.

Points for attention

1. Palo is unable to guarantee the internal sequence of a batch of loaded data, so it is necessary to cooperate with Sequence Column to ensure data sequence in scenarios such as CDC.

Sequence-Column

In UNIQUE KEY mode, Palo will automatically update the data according to the primary key. When rows with the same primary key appear in the same batch of loaded data, however, Palo can't judge their sequence, which can lead to inconsistent update behavior.

In some data synchronization scenarios, it is necessary to ensure that the data can be updated in order, which is the function of Sequence Column.

Implementation principle

Sequence Column only supports tables of UNIQUE KEY model. The principle is to add a hidden column `__DORIS_SEQUENCE_COL__` to the table. The user needs to specify the column type when creating the table.

In the loaded source data, the user needs to add an additional sequence column whose type `__DORIS_SEQUENCE_COL__` is specified when creating the table. Palo will determine the order of the data and update according to the value of the sequence column in this order.

Enable Sequence Column function

This is a new function introduced after Palo 3.10.

1. Create a new table

   Set Sequence Column as follows when creating a table:

```
CREATE TABLE order_table
(
    order_id BIGINT,
    order_type VARCHAR(8),
    order_status VARCHAR(32)
)
UNIQUE KEY(order_id)
DISTRIBUTED BY HASH(order_id) BUCKETS 8
PROPERTIES
(
    "function_column.sequence_type" = 'Date'
);
```

   Here, we specify Sequence Column type in `PROPERTIES` as Date to enable this function. Refer to CREATE TABLE command manual for more instructions.

2. Enable this function for old tables

   For tables of UNIQUE KEY created before version 3.10, enable this function through the following commands:

```
ALTER TABLE order_table ENABLE FEATURE "SEQUENCE_LOAD"
WITH PROPERTIES ("function_column.sequence_type" = "Date")
```

This operation, in essence, is a Schema Change operation, view the job execution progress through SHOW ALTER TABLE COLUMN after execution.

Set a variable to show hidden columns to determine whether a table has enabled Mark deletion function.

```
SET show_hidden_columns=true`
```

Then use `DESC tablename`. If there is `__DORIS_SEQUENCE_COL__`column in output, it indicates that the function has been enabled in the table.

Use sequence column function in the load

The methods used in different data load modes are slightly different. This function currently supports the following data load modes:

- STREAM LOAD
- BROKER LOAD
- ROUTINE LOAD

Refer to the respective documents for specific syntax. Here, we only give simple examples of different load methods. Suppose that the original loaded data are as follows:

```
1000,TYPE#1,PENDING,2020-10-01
1001,TYPE#2,PAID,2020-10-02
1002,TYPE#3,PENDING,2020-10-03
1001,TYPE#2,PENDING,2020-10-01
1004,TYPE#3,PAID,2020-10-03
```

Stream Load

```
curl --location-trusted -u root \
-H "columns: order_id, order_type, order_status, source_sequence"
-H "function_column.sequence_col: source_sequence" \
-T data.txt http://host:port/api/example_db/order/_stream_load
```

We name the fourth column `source_sequence` in `columns` attribute of Header, then set the column as sequence column in attribute `function_column.sequence_col`.

Thus, the final status of the order 1001 in the source data will be `PAID`.

Broker Load

```
LOAD LABEL example_db.label1
(
    DATA INFILE("hdfs://host:port/user/data/*/test.txt")
    INTO TABLE `order`
    COLUMNS TERMINATED BY ","
    (order_id, order_type, order_status, source_sequence)
    ORDER BY source_sequence
)
WITH BROKER 'bos'
(
    ...
);
```

Configure the sequence column by clause `ORDER BY`.

routine load

```
 CREATE ROUTINE LOAD example_db.job_name ON order
COLUMNS(order_id, order_type, order_status, source_sequence),
ORDER BY source_sequence
PROPERTIES
(
    ...
)
FROM KAFKA
(
    ...
);
```

Configure the sequence column by clause `ORDER BY`.

Usage examples

Here take Stream Load as an example to show the use method and effect of sequence column through an actual example.

1. Create a table that supports Sequence Column

```
 CREATE TABLE test_table
(
    user_id     BIGINT,
    date        DATE,
    group_id    BIGINT,
    keyword     VARCHAR(128)
)
UNIQUE KEY(user_id, date, group_id)
DISTRIBUTED BY HASH(user_id, date) BUCKETS 10
PROPERTIES
(
    "function_column.sequence_type" = 'Date'
)
```

Then we can view the hidden columns:

```
mysql> set show_hidden_columns=true;
Query OK, 0 rows affected (0.00 sec)
mysql> desc test_table;
+-----------------------+-------------+------+-------+---------+---------+
| Field                 | Type        | Null | Key   | Default | Extra   |
+-----------------------+-------------+------+-------+---------+---------+
| user_id               | BIGINT      | Yes  | true  | NULL    |         |
| date                  | DATE        | Yes  | true  | NULL    |         |
| group_id              | BIGINT      | Yes  | true  | NULL    |         |
| keyword               | VARCHAR(128) | Yes  | false | NULL    | REPLACE |
| __DORIS_SEQUENCE_COL__ | DATE        | Yes  | false | NULL    | REPLACE |
+-----------------------+-------------+------+-------+---------+---------+
5 rows in set (0.00 sec)
```

2. Load data normally

   Load the following data:

```
1,2020-02-22,1,2020-02-22,a
1,2020-02-22,1,2020-02-22,b
1,2020-02-22,1,2020-03-05,c
1,2020-02-22,1,2020-02-26,d
1,2020-02-22,1,2020-02-22,e
1,2020-02-22,1,2020-02-22,b
```

   Map the Sequence Column as the fourth column in the source data, column `modify_date`.

```
curl --location-trusted -u root: \
-H "column_separator:," \
-H "columns: user_id, date, group_id, modify_date, keyword" \
-H "function_column.sequence_col: modify_date" \
-T testData http://host:port/api/test/test_table/_stream_load
```

   The result is:

```
mysql> select * from test_table;
+---------+------------+----------+---------+
| user_id | date       | group_id | keyword |
+---------+------------+----------+---------+
|       1 | 2020-02-22 |        1 | c       |
+---------+------------+----------+---------+
```

Also, we can view the values of hidden columns:

```
mysql> set show_hidden_columns=true;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from test_table;
+---------+------------+----------+---------+----------------------+
| user_id | date       | group_id | keyword | __DORIS_SEQUENCE_COL__ |
+---------+------------+----------+---------+----------------------+
|       1 | 2020-02-22 |        1 | c       | 2020-03-05           |
+---------+------------+----------+---------+----------------------+
```

In this load, because `2020-03-05` is the maximum value in the value of Sequence Column (the value of `modify_date`), `c` is finally retained in the `keyword` column.

3. Guarantee of replacement sequence

   Impot the following data after the above steps are completed:

```
1,2020-02-22,1,2020-02-22,a
1,2020-02-22,1,2020-02-23,b
```

   Query data

```
MySQL [test]> select * from test_table;
+---------+------------+----------+---------+
| user_id | date       | group_id | keyword |
+---------+------------+----------+---------+
|       1 | 2020-02-22 |        1 | c       |
+---------+------------+----------+---------+
```

   No replacement occurs because the Sequence Column of the newly-loaded data are all smaller than the existing values in the table.

Then try loading the following data:

```
  1,2020-02-22,1,2020-02-22,a
1,2020-02-22,1,2020-03-23,w
```

Query data:

```
 MySQL [test]> select * from test_table;
+---------+------------+----------+---------+
| user_id | date       | group_id | keyword |
+---------+------------+----------+---------+
|       1 | 2020-02-22 |        1 | w       |
+---------+------------+----------+---------+
```

The data are replaced because the Sequence Column value of the newly-loaded data is greater than the value in the table.

Data Export

Export Overview

Users, in some cases, would like to export the data in Palo to other systems for further analysis.

Palo currently supports the following ways to export data:

1. Export

   EXPORT command mainly exports the contents of the whole table (or specified section) to remote storage, such as BOS.

   This method currently supports only the full export of specified tables or sections rather than the mapping, filtering or converting of exported results. The export format is CSV.

   Refer to the document for details: Full data export.

2. SELECT OUTFILE

   SELECT OUTFILE command exports user's SQL query results to remote storage (BOS) or local directory.

   This method can acquire expected results through flexible SQL syntax with lower efficiency than Export method due to the limitation of single thread output of the final result, though.

   Refer to the document for details: Export query result set.

3. Export to external table

   Users can, through INSERT command, export data query results to external tables.

   Refer to the document for details: Export data to external table.

Full Data Export

Full data export (Export) is a data export function provided by Palo.

This function can export the table or section data, specified by the user, to remote storage as HDFS / BOS through Broker process in text format.

This document mainly introduces the basic use of Export function.

Function introduction

Export function is asynchronous. The system generates a distributed data scanning plan after the user specifies the table or some sections in the table to be exported through Export statement, and multiple Compute Nodes scan and read the data to write them to the remote storage through Broker process.

The minimum granularity of Export function is the section of the table.

Currently the export function is rather simple, supports only the whole column export of the table rather than the mapping, filtering or converting of the columns of the table.

Submit export operation

Submit an export operation with the following statement

```
 EXPORT TABLE example_tbl
PARTITION(p1, p2)
TO "bos://my_bucket/export/"
WITH BROKER "bos"
(
    "bos_endpoint" = "http://bj.bcebos.com",
    "bos_accesskey" = "xxxxxxxxxxxxxxxxxxxxxxxxx",
    "bos_secret_accesskey"="yyyyyyyyyyyyyyyyyy"
);
```

The statement specifies to export section `p1` and `p2` in table `example_tb1` to the directory `bos://my_bucket/export/` under BOS.

Refer to EXPORT for detailed help of export command.

Execution of export operation

The export operation generates a host of query plans, each of which is responsible for scanning partial data tablet (Tablet).

Each query plan scans 5 tablets by default. That is, assuming there are 100 Tablets in total, 20 query plans will be generated.

Also, the user can specify the value through operation property `tablet_num_per_task` when submitting the operation.

Multiple query plans of one operation are executed sequentially.

A query plan scans a host of Tablets and organizes the read data in the form of rows, and then write the data to the remote storage through calling Broker with every 1024 rows seen as a batch.

The query plan will automatically retry 3 times if it encounters any error. If a query plan still fails after 3 retries, the whole operation fails.

Structure of exported file

First, export operation creates a temporary directory called `__doris_export_tmp_12345`(where 12345 is the operation id) in

the specified remote storage path.

The exported data are first written to the temporary directory. Each query plan generates a file, whose name is shown as follows as examples:

```
export-data-c69fcf2b6db5420f-a96b94c1ff8bccef-1561453713822
```

where `c69fcf2b6db5420f-a96b94c1ff8bccef` is the ID of the query plan. `1561453713822` is the timestamp of the file generation.

Doris will transfer these files to the path specified by the user after the export of all the data.

View operation progress

The user can query the imported operation status after submitting the operation through SHOW EXPORT command. The results are as follows:

```
     JobId: 14008
     State: FINISHED
  Progress: 100%
  TaskInfo: {"partitions":["p1", "p2"],"exec mem limit":2147483648,"column separator":",","line delimiter":"\n","tablet num":1,"broker":"hdfs","coord num":1,"db":"default_cluster:db1","tbl":"tbl3"}
      Path: bos://my_bucket/export/
CreateTime: 2019-06-25 17:08:24
 StartTime: 2019-06-25 17:08:28
FinishTime: 2019-06-25 17:08:34
   Timeout: 3600
  ErrorMsg: N/A
```

The export is complete when the operation status showed FINISHED.

Refer to : SHOW EXPORT for detailed help about SHOW EXPORT.

Export Data to External Table

As a way of data export, Palo supports writing data directly to ODBC external tables through `INSERT` command.

Create an ODBC external table through CREATE ODBC TABLE first.

Then write the data to the external table through the following commands:

```
INSERT INTO extern_tbl VALUES(1, 2, 3);

INSERT INTO extern_tbl SELECT * FROM other_tbl;
```

It is not recommended to submit a large number of data writings at one time for the writing operation is made in a single ODBC Client connection mode.

Also, Palo supports write transaction for ODBC external tables. Execute `INSERT` command by setting the session variable `enable_odbc_transcation` to start transaction support:

```
SET enable_odbc_transcation = true;
INSERT INTO extern_tbl SELECT * FROM other_table;
```

Transaction support can ensure the atomicity of data writing to avoid partial data writing.

Please start write transaction as appropriate for it will reduce write efficiency.

Refer to INSERT for more information about INSERT command.

Export Query Results Set

This document teaches you how to export query results by using SELECT INTO OUTFILE command .

Function introduction

`SELECT INTO OUTFILE` statements are able to export the query results to a file.

Currently you can use Broker process to export the query results into remote storage, such as HDFS, S3 and BOS. Or the results can be exported directly to local disk of the node where the Compute Node is located (not available for cloud users).

The user can export the expected query results by the command together with flexible SQL syntax.

Execute export command

`SELECT INTO OUTFILE`, in essence, is a synchronous SQL query command, which means it will be affected by timeout limit of the session variables `query_ timeout`. Please set a reasonable timeout in advance if relatively large result set is exported or if there is a relatively long time to execute SQL.

1. Export to BOS

```
SELECT * FROM example_tbl
INTO OUTFILE "bos://my_bucket/result_"
FORMAT AS CSV
PROPERTIES
(
    "broker.name" = "bos",
    "broker.bos_endpoint" = "http://bj.bcebos.com",
    "broker.bos_accesskey" = "xxxxxxxxxxxxxxxxxxxxxxxxx",
    "broker.bos_secret_accesskey" = "yyyyyyyyyyyyyyyyyyyyyyyy"
    "column_separator" = ",",
    "line_delimiter" = "\n",
    "max_file_size" = "100MB"
);
```

The above commands export the results of SQL statement `SELECT * FROM example_tbl` to BOS.

2. Export to local disk of Compute Node

```
SELECT * FROM example_tbl
INTO OUTFILE "file:///home/work/path/result_"
FORMAT AS CSV
PROPERTIES
(
    "column_separator" = ",",
    "line_delimiter" = "\n",
    "max_file_size" = "100MB"
);
```

The above commands export the results to specified disk path of certain Compute Node.

Public cloud Palo users cannot export the results to local disk for they have no direct access to the node.

3. Export to BOS and generate an identification file after successful export.

```
SELECT * FROM example_tbl
INTO OUTFILE "bos://my_bucket/result_"
FORMAT AS CSV
PROPERTIES
(
    "broker.name" = "bos",
    "broker.bos_endpoint" = "http://bj.bcebos.com",
    "broker.bos_accesskey" = "xxxxxxxxxxxxxxxxxxxxxxxxx",
    "broker.bos_secret_accesskey" = "yyyyyyyyyyyyyyyyyyyyyyyy"
    "column_separator" = ",",
    "line_delimiter" = "\n",
    "max_file_size" = "100MB",
    "success_file_name" = "SUCCESS"
);
```

The above commands export the results of SQL statement `SELECT * FROM example_tbl` to BOS and generate an empty file identification of `result_SUCCESS` after successful export. The user can judge whether the export is completed through this identification.

Please refer to SELECT INTO OUTFILE for a detailed description of the statement.

View returned results

Export commands are synchronous. The return of the commands indicates the end of the operation. And a row of results will be returned to show the execution result of the export.

For normal export and return, the results are as follows:

```
mysql> select * from tbl1 limit 10 into outfile "file:///home/work/path/result_";
+------------+-----------+----------+-------------+
| FileNumber | TotalRows | FileSize | URL         |
+------------+-----------+----------+-------------+
|          1 |         2 |        8 | 192.168.1.10 |
+------------+-----------+----------+-------------+
1 row in set (0.05 sec)
```

- FileNumber: Number of finally generated files.
- TotalRows: Row number of result set.
- FileSize: Total exported file size. Unit byte.

- URL: If the results are exported to a local disk, the specific Compute Node to which it is exported will be displayed here.

For execution errors, error messages will be returned, such as:

```
mysql> SELECT * FROM tbl INTO OUTFILE ...
ERROR 1064 (HY000): errCode = 2, detailMessage = Open broker writer failed ...
```

BI Tool Access

Sugar

Launched by Baidu smart cloud, Sugar , rich in components and out-of-the-box, is an agile BI and data visualization platform, which needs no SQL and any encoding. Palo can be compatible with the cloud visualization platform Sugar perfectly. Cooperating with Sugar, Palo can achieve high-performance visual chart analysis and powerful interactive analysis.

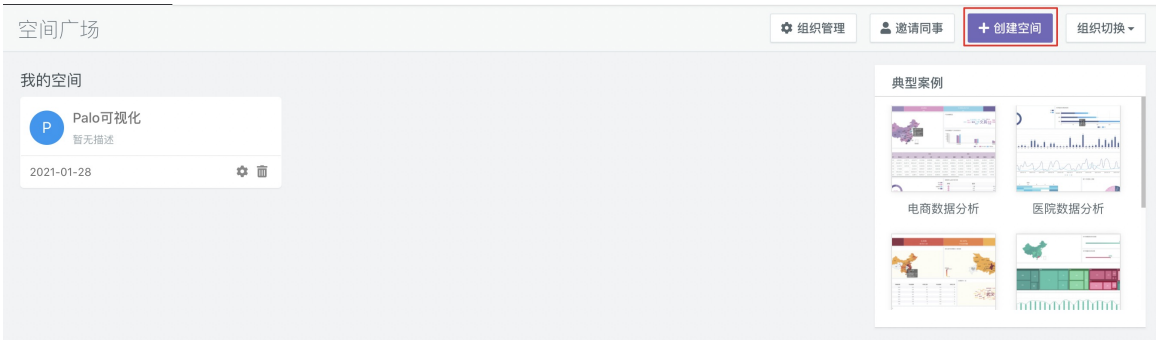
You can connect your visualization tool Sugar according to the following steps.

Preparations

- Open Sugar service.
- Install MySQL JDBC driver.

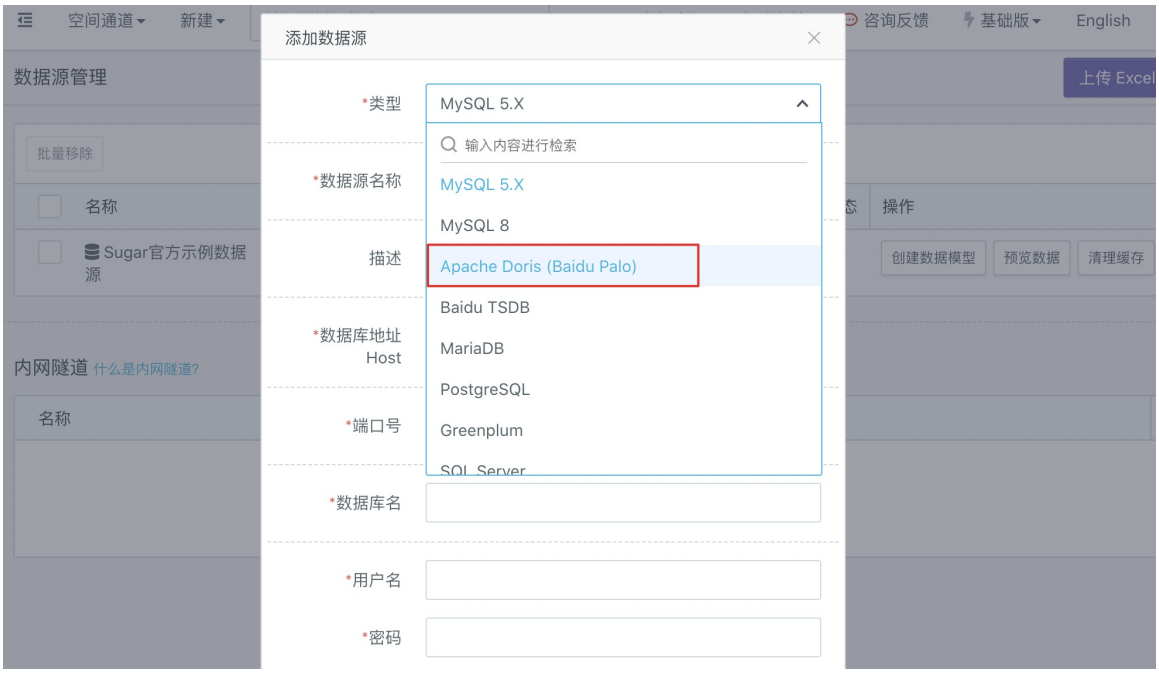
Create Palo data source in Sugar

1. Enter the Sugar service to create a zone on the home page, or use the created zone.



2. Click "Data source" on the left after entering the created zone, and click the button "Add data source" in the upper left corner of the page.

3. Select "Apache Doris (Baidu Palo)" from "Type" option box.



4. Fill in the connection information of Palo cluster on the page of "Add data source".

Configuration items	Instructions for filling
Data source name	Create a data source name for subsequent management
Description	Self-defined descriptions are suggested to supplement the detailed management information for the data source
Database address Host	Bind the LeaderNode with EiP on the page of Palo cluster details - Configuration information and fill in this IP address (Eip needs to be purchased and bound by the user).
Port	9030
Database name	Database name in Palo
User name	The account created in Palo cluster, the administrator account is "admin".
Password	The connection password when creating the cluster.

5. Click "Connection test" below for connectivity test after filling in the configuration information, then click "Add" button on the page when the system prompts "Connected". In this way, Sugar has been successfully connected to the Palo database.

Use Sugar

1. Data analysis for Palo can be achieved in Sugar after completed data source connection. The added data source can be seen in data source management list.

2. Then various operations for the data source, such as "Create data model", can be done by clicking to perform visual data analysis in corresponding page.

Refer to [Sugar help document] for more instructions on using Sugar. (https://cloud.baidu.com/doc/SUGAR/index.html).

Navicat

Palo supports the connection to Navicat, you can connect your database management tool Navicat according to the following steps.

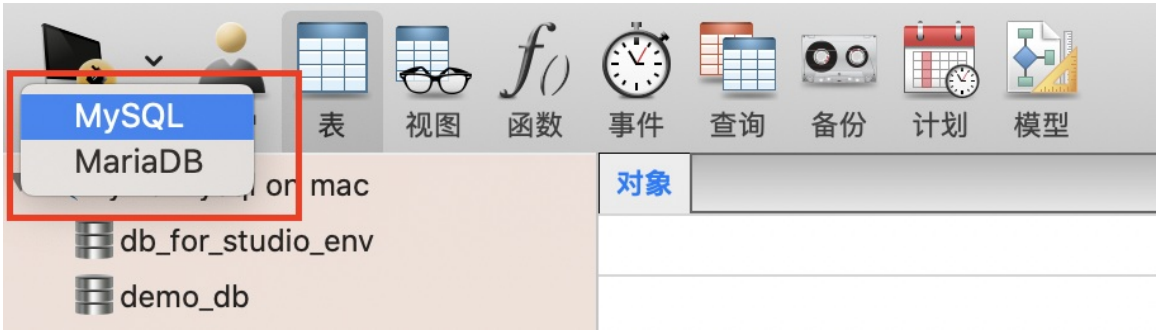
Preparations

Install Navicat.

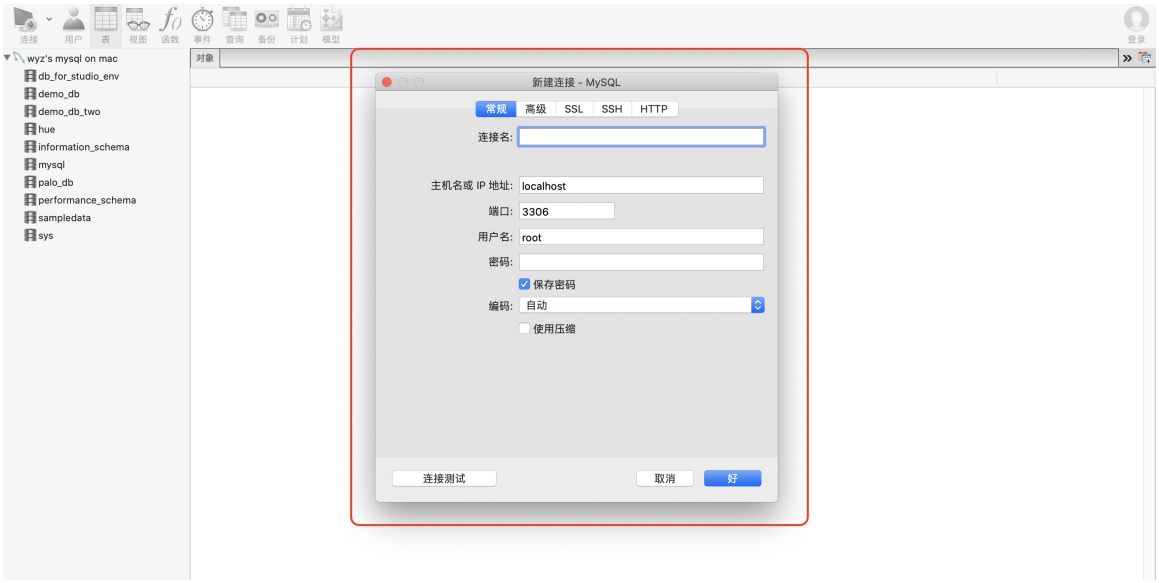
Bind EIP for Leader Node of Palo cluster.

Connect Palo

1. Create a new connection and select **MySQL**



2. Then configure the connection information：



- **Connection name**: User-defined name
- **IP address**: Fill in the EIP binding address of Palo Leader Node (LeaderNode protocol public address)
- **Port**: Fill in the port number of the **MySQL** protocol connection target, generally defaulted as 9030.

- **User name**: The account created in Palo cluster, the administrator account is "admin".
- **Password**: The connection password when creating the cluster.

3. Click **Connection test** after configuration. The display of **Connected** indicates that Navicat can successfully access Palo .

4. Finally, click **OK** to manage and use **Public Cloud Palo cluster** with Navicat.

Tableau

Palo supports the connection to Tableau, you can connect your Tableau according to the following steps.

Preparations

Install Tableau.

Bind EIP for Leader Node of Palo cluster.

Connect Palo

1. Create a new file in an arbitrary place: **mysql.tdc** and add the following contents to the file:

```
<connection-customization class='mysql' enabled='true' version='2019.4'>
<vendor name='mysql' />
<driver name='mysql' />
<customizations>
   <customization name='odbc-connect-string-extras' value='DEFAULT_AUTH=mysql_native_password'/>
</customizations>
</connection-customization>
```

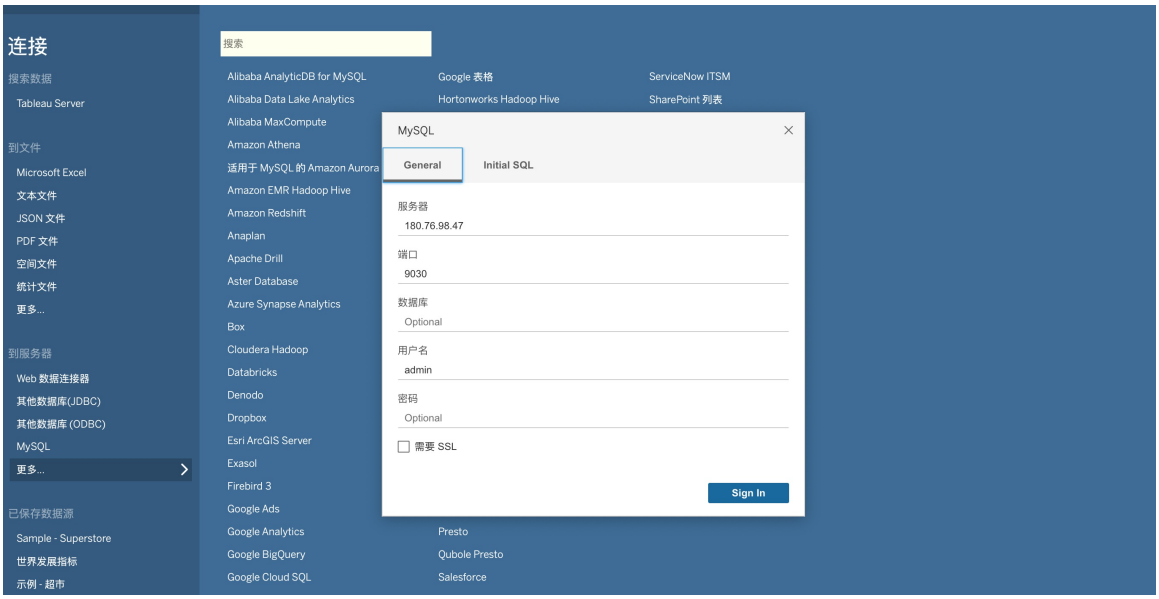
   Then save the file. The file is used for configuring MySQL ODBC Driver, and the default authentication protocol is `mysql_native_password`.

   Under Windows system: go to the directory `C:\Users\<user>\Documents\My Tableau Repository\Datasources` to transfer `mysql.tdc` to this directory.

   Under Mac system: go to the directory : `/Users/wangyouzhuo/Documents/My Tableau Repository/Datasources` to transfer `mysql.tdc` to this directory.

2. Restart **Tableau** , click **More** and select **MySQL**.

Just fill in **IP / Host / User name / Password**, other items are unnecessary.



- **Server**: Fill in the EIP address of Palo Leader Node (LeaderNode protocol public address)
- **Port**: Fill in the port number of the **MySQL** protocol connection target, generally defaulted as 9030.

- **Database**： no need to fill in.
- **User name**： The account created in Palo cluster, the administrator account is "admin".
- **Password**： The connection password when creating the cluster.

3. Click **Sign in** to log in.

DBeaver

Palo supports the connection to DBeaver, you can connect your database management tool DBeaver according to the following steps.

Preparations

Install DBeaver.

Bind EIP for Leader Node of Palo cluster.

Connect Palo

1. Create a new connection and select **MySQL** , then click **Next step**

2. Then configure the connection information:

- **Server address**: Fill in the EIP binding address of Palo Leader Node (LeaderNode protocol public address)
- **Port**: Fill in the port number of the **MySQL** protocol connection target, generally defaulted as 9030.

- **Database**: It is recommended not to fill in or fill in any database name in the Palo cluster. Filling in other strings may cause failed connection.
- **User name**: The account created in Palo cluster, the administrator account is "admin".
- **Password**: The connection password when creating the cluster.

3. No need to fill in other information. Click **Test link** after configuration,. The display of **Connected** indicates that DBeaver can successfully access Palo .

4. Click **Done** to manage and use **Public Cloud Palo cluster** with DBeaver.

YONGHONGBI

Palo supports the connection to Yonghong BI, you can connect your Yonghong BI according to the following steps.

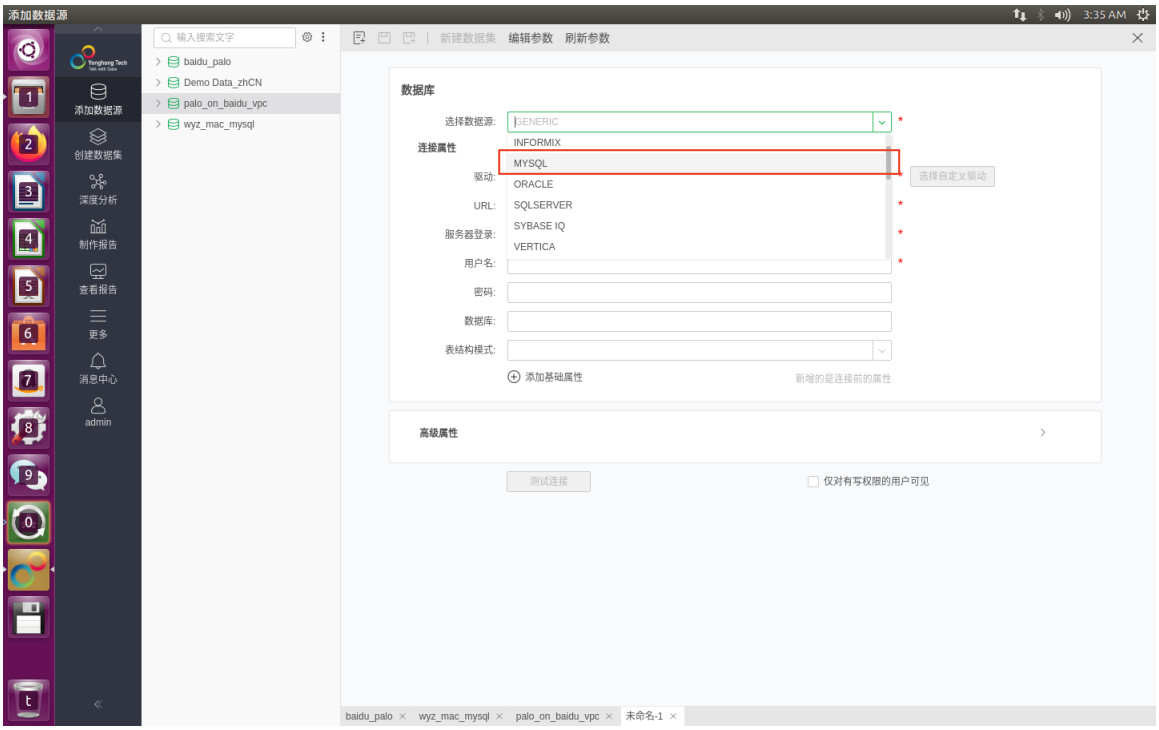
Preparations

Install and log in Yonghong BI.

Bind EIP for Leader Node of Palo cluster.

Connect Palo

1. Start **Yonghong BI**, click **New** button on the top left and select **MySQL** at **Connect data source**.

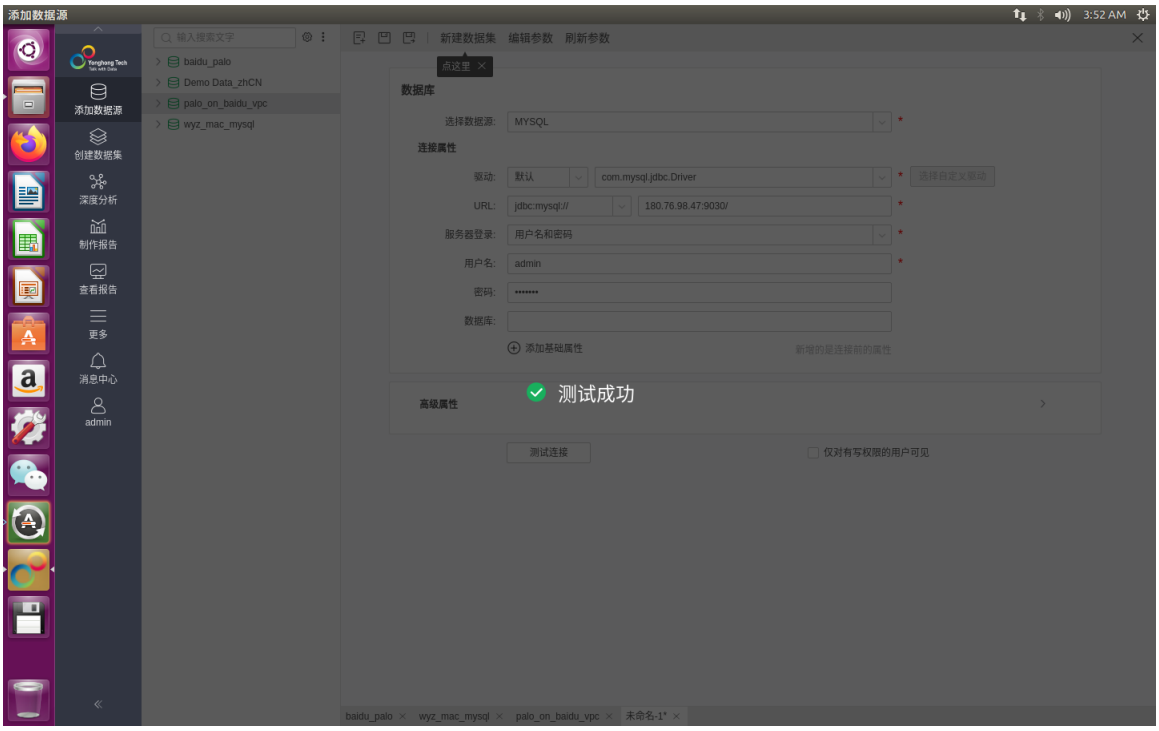


2. Then fill in the data source configuration as the following ways:

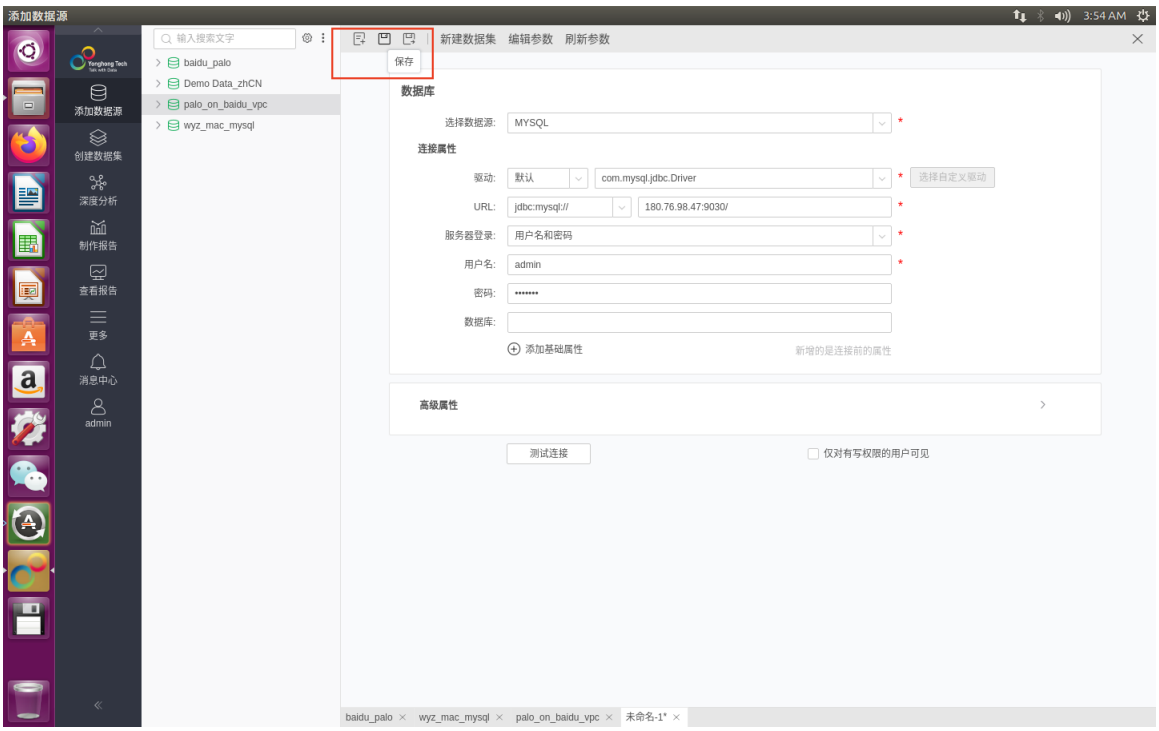
   - **Driver**: select **Default driver: com.mysql.jdbc.Driver**
   - **URL**: **jdbc:mysql://Databaseserver:port/DatabaseName**. Fill in Leader Node binding EIP of Palo cluster at **Databaseserver**; Fill in the port number of the **MySQL** protocol connection target of Palo cluster at **port**, generally defaulted as 9030, as shown in the figure below; **DatabaseName** is unnecessary to fill in.

   - **Server login**: Select **User name and password**
   - **User name**: The account created in Palo cluster, the administrator account is **admin**.
   - **Password**: The connection password when creating the cluster.
   - Other items are unnecessary to fill in.

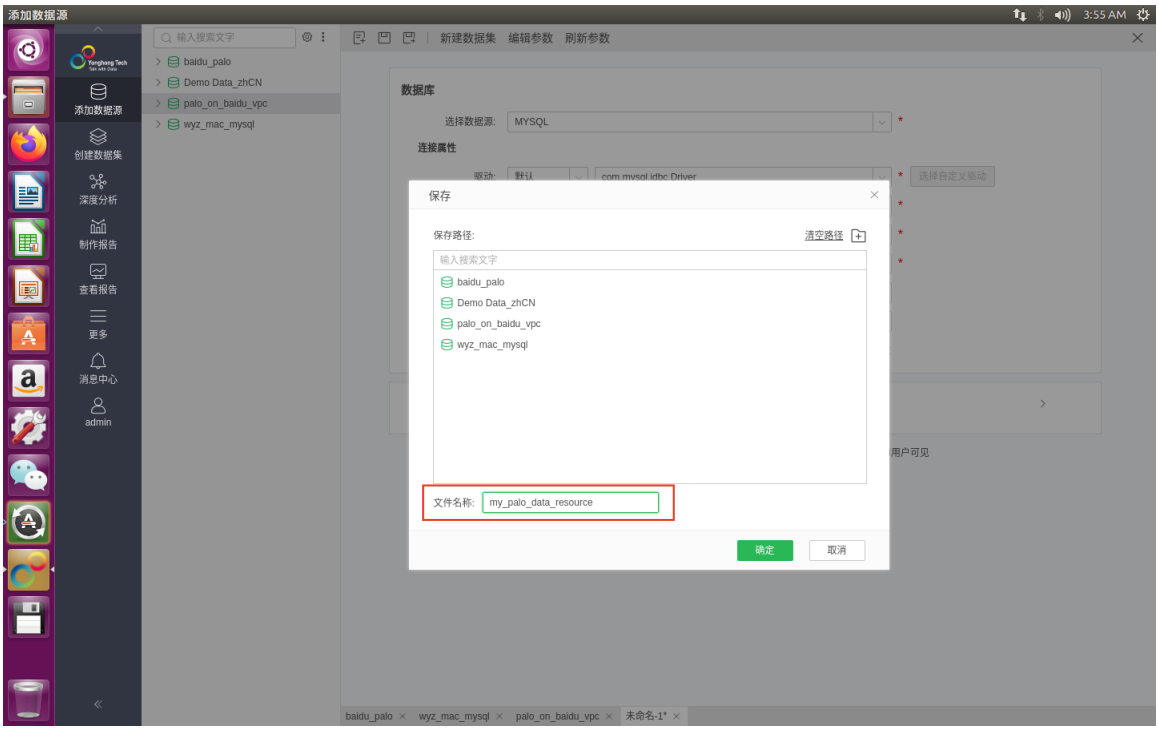
3. Click **Connection test**, the following display indicates that the connection is succeeded.



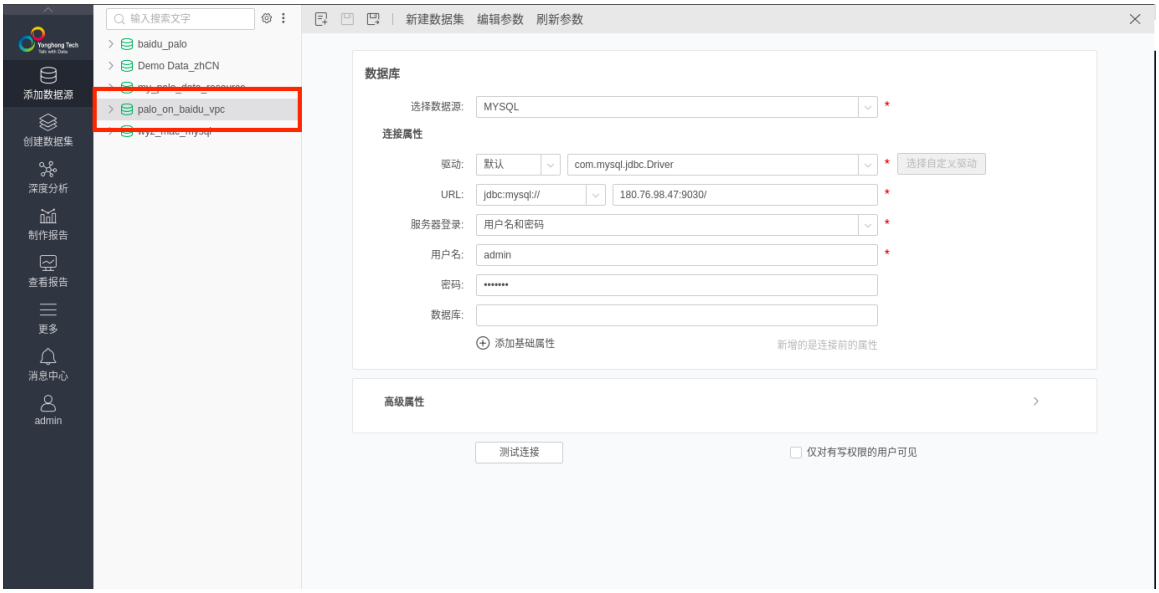
Then click **Save**



Fill in the data source name (defined by the user) at **File name**, then click **OK** to save the data source.



Next, you can see the data source you just added in the directory tree on the left.



Thus, we have connected **Palo data source** to **Yonghong BI** to perform free exploration and data analysis.

FineBI

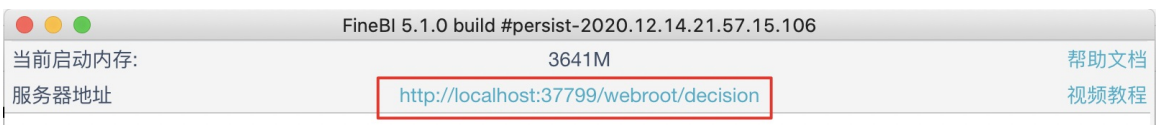
Palo supports connecting to FineBI. You can follow these instructional steps to connect to your visualization tool, FineBI.

Preparation

- Install FineBI version 5.0 and above.
- Install MySQL JDBC Driver.

Log in to the server

1. Open FineBI , click on the "Server Address" in the software to configure your account.



2. Configure your account on the server page opened.

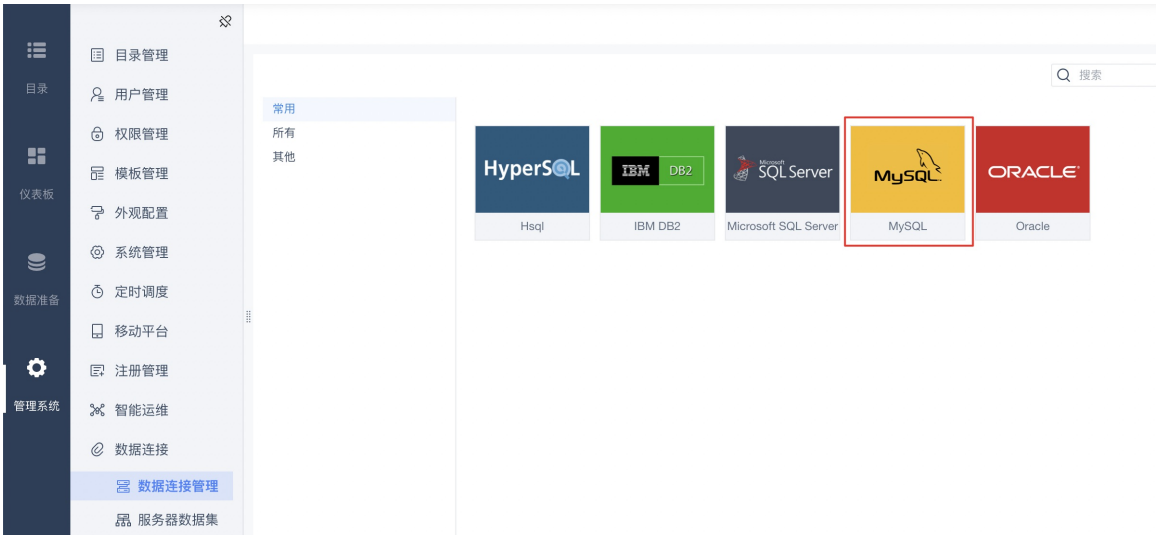
3. After successful configuration of your account, go to the next step: Database Selection.

4.Select the database for configuration. For corporate formal usage, please refer to FineBI Help Document.Configuration of External Databasefor configuration. Here for easier demonstration, we will choose the "Internal Database" and log in directly.

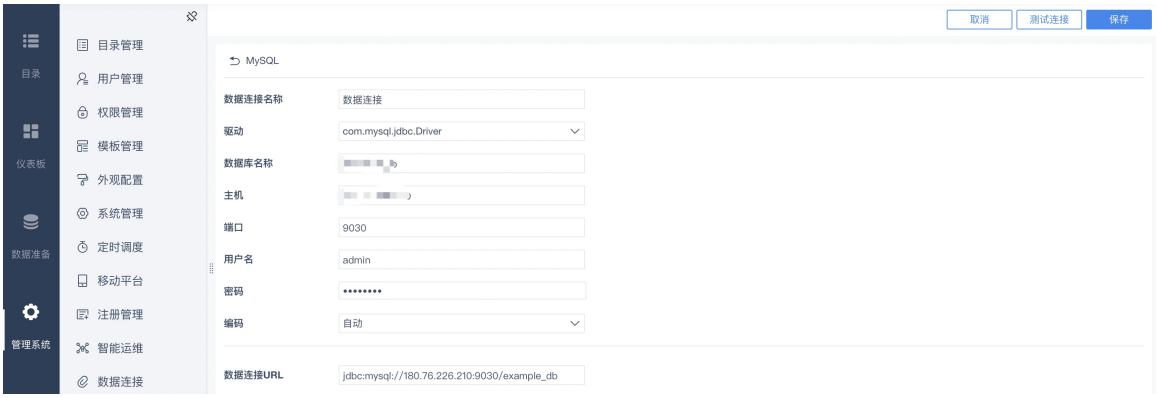
5.Log in and enter FineBI.

Connect to Palo

1.Click on "System Management - Data Connection - Data Connection Management", and choose the MySQL Type.

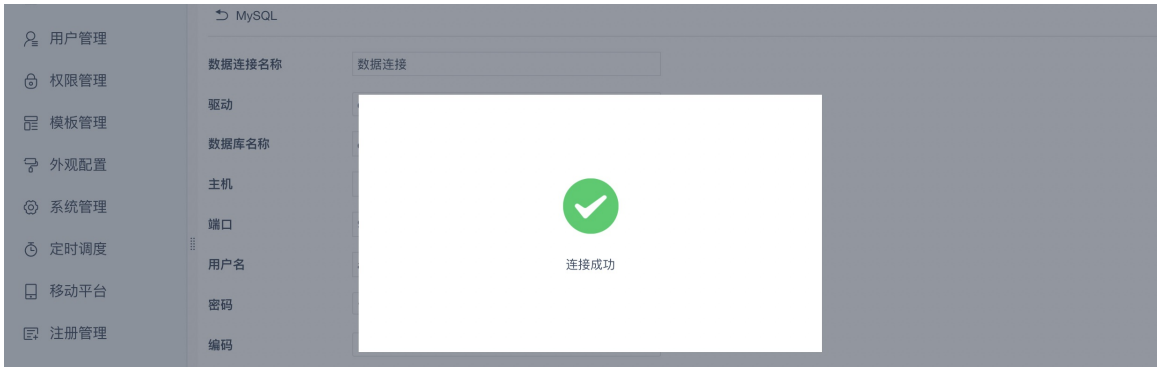


2.On the page of Database Configuration, fill in the connection information of Palo Cluster.



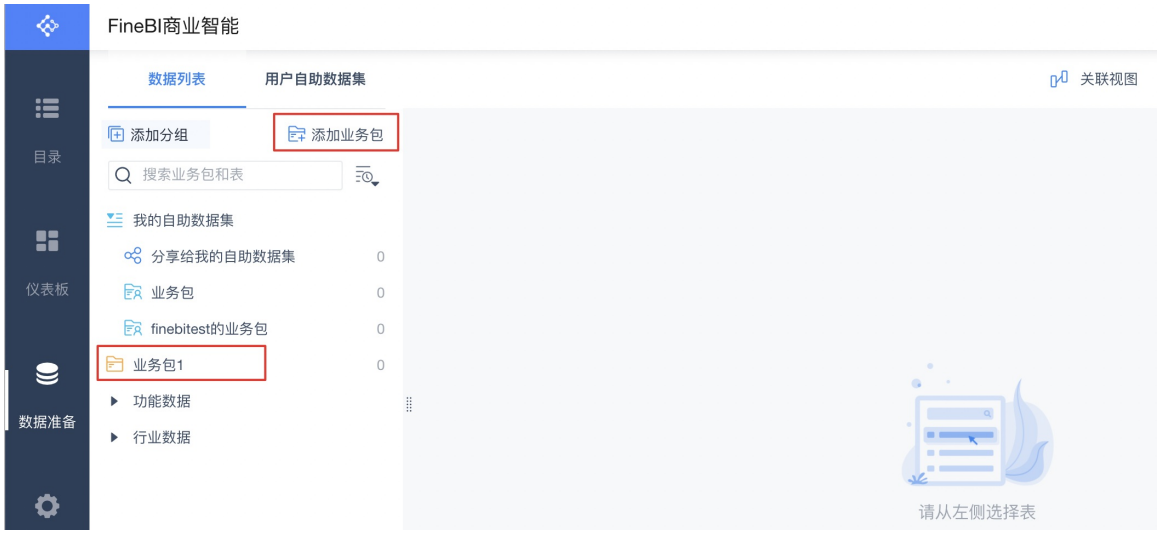
Configuration Item	Filling Instruction
Name of Data Connection	Create a name for the connection for easier future management
Driver	Select`com.mysql.jdbc.Driver`
Name of Database	Name of the Database in Palo
Host	On the page of the Palo Cluster - Details - Configuration Information, bind Eip for LeaderNode and fill in this IP address
Port	9030
User Name	Account created in Palo Cluster, administrator's account admin
Password	Connection password created when creating the cluster

3.After filling Configuration information, click "Connection Test" on the top right corner to test the connection. When the system prompts success connection, click "Save" on the top right corner of the page. At this point, FineBI has been successfully connected to the Palo Database.



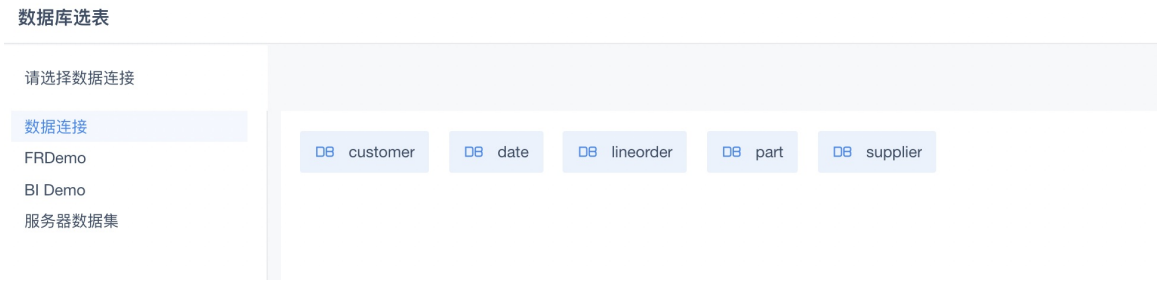
Use FineBI

1. After successful connection of the database, Palo data analysis in FineBI is available. Click on "Data Preparation" to add business package.

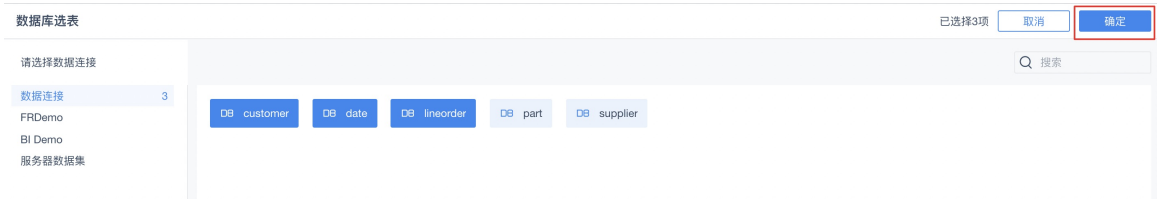


2.Enter the business package, click on "Add Table" button, select the database table and enter.

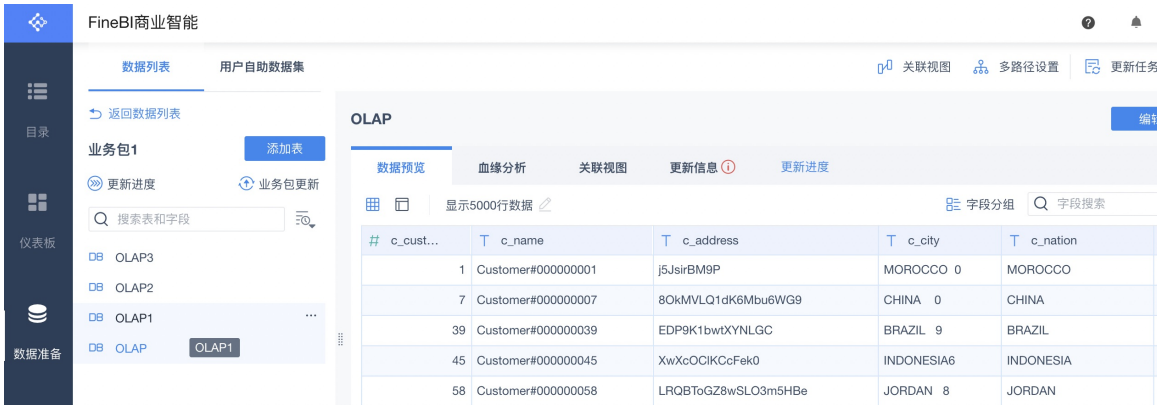
3.Already connected tables in the Palo database are visible on the page of database tables.



4.Selected the data to be analyzed, click "confirm" to start database connection.



5.After successful database connection, FineBI is ready for visual analysis of the data.



For more instructions on how to use FineBI, please refer to [FineBI Help Document]. (https://help.fanruan.com/finebi/)

Backup and Recovery

The backup and restoring function is mainly used to quickly back up the cluster snapshot to the remote storage and quickly recover from the backup data when needed.

The backup function, unlike Data export function, is better than the export function in terms of overall speed because it directly copies the data file to the remote storage, however, the backup data can only be used for the **restoring** function of Palo itself, and the **exported** data can be read and used by other systems.

Basic Concepts

1. Repository

   The user needs to create a **Repository**, which is a mapping of a directory on a remote storage system to Palo, before backup and restoring. The backup operation uploads data to this path, while the restoring operation downloads data from this path.

   Palo supports the creation and deletion of repository. Refer to CREATE REPOSITORY and DROP REPOSITORY command manual for specific help. View the created repository through SHOW REPOSITORIES command.

2. Backup

   The backup operation, with minimum partition granularity, can be directly uploaded to the remote repository for storage in the form of Palo files. The system will perform the following operations after the user submits a backup request:

   1. Snapshot and its upload

      Specified tables or partition data file can be snapshotted during snapshot phase. After that, the backup will be performed for snapshot. After the snapshot, changes and import of the table and other operations will no longer affect the results of the backup. Snapshot is just a hard chain for the current data file, which takes little time. After the completion of snapshot, the snapshot files will be uploaded one by one, which is concurrently completed by each Compute Node.

   2. Preparation and upload of metadata

      After the data file snapshot is uploaded, Leader Node will first write the corresponding metadata as a local file, and then upload the local metadata file to the remote repository to finish final backup job.

   Refer to BACKUP syntax manual for specific operations for backup. The backup operation is an asynchronous operation, whose progress can be viewed through SHOW BACKUP command. At the same time, it can also cancel a running backup operation through CANCEL BACKUP command.

3. Restore

   The restoring operation needs to specify a pre-existing backup in the remote repository, and then restores backup contents to the local cluster. The system will perform the following operations when the user submits a Restore request:

   1. Create the corresponding metadata locally

      In this step, the corresponding table partitions and other structures will be created and restored in the local cluster. After creation, the table is visible but not accessible.

   2. Download snapshot

      Download the snapshot file from the remote repository to the corresponding Compute Node.

   3. Effective snapshot

      After downloading the snapshot, we need to map each snapshot to the metadata of the current local table. Then reload these snapshots to put them into effect and complete final restoring job.

   Refer to RESTORE syntax manual for specific restoring operations. The restoring operation is an asynchronous operation, whose progress can be viewed through SHOW-RESTORE command. Also, a running restoring operation can be canceled through CANCEL RESTORE command.

Operation Examples

With a complete example, Let's show how to migrate cluster A data to cluster B through backup and restoring operation.

1. Create a repository in cluster A

```
CREATE REPOSITORY `bos_repo`
WITH BROKER `bos`
ON LOCATION "bos://my_bucket/palo_backup"
PROPERTIES
(
    "bos_endpoint" = "http://bj.bcebos.com",
    "bos_accesskey" = "xxxxxxxxxxxxxxxxx",
    "bos_secret_accesskey"="yyyyyyyyyyyyyy"
);
```

Create a repository with the name of `bos_repo`, pointing to `palo_backup` directory. Refer to CREATE REPOSITORY for more detailed help.

2. Backup data in cluster A

```
BACKUP SNAPSHOT example_db.snapshot1
TO `bos_repo`
ON
(
    example_tbl PARTITION (p1,p2),
    example_tbl2
);
```

Specify two partitions of table in `example_tbl` database `example_db` and table `example_tbl2` for backup and then back them up to repository `bos_repo` with the backup name being `snapshot1` this time. Refer to BACKUP for more detailed help of backup operation.

The backup operation is an asynchronous operation, whose progress can be viewed through SHOW BACKUP command. When the field `State` in returned result is FINISHED, the backup is completed.

3. Create an identical repository in cluster B:

```
 CREATE REPOSITORY `bos_repo`
WITH BROKER `bos`
ON LOCATION "bos://my_bucket/palo_backup"
PROPERTIES
(
    "bos_endpoint" = "http://bj.bcebos.com",
    "bos_accesskey" = "xxxxxxxxxxxxxxxxx",
    "bos_secret_accesskey"="yyyyyyyyyyyyy"
);
```

4. View the backup snapshot of the repository in cluster B

```
 SHOW SNAPSHOT ON `bos_repo`;
```

Refer to SHOW SNAPSHOT syntax manual for more help.

5. Restore data in cluster B

```
 RESTORE SNAPSHOT example_db.`snapshot1`
FROM `bos_repo`
ON
(
    `example_tbl2`
)
PROPERTIES
(
    "backup_timestamp"="2020-05-04-16-45-08",
    "replication_num" = "1"
);
```

Specify backup data called `snapshot1` in `bos_repo` to select to restore its table `example_tbl2`.

Each backup data has a timestamp（`backup_timestamp`）, which needs to be specified to show. Here, we specify to restore only one copy.

The restoring operation is also an asynchronous operation, whose specific progress can be viewed through SHOW RESTORE command. When the field `State` in returned result is FINISHED, the restoring is completed.

Best Practice

Backup

Currently, we support full backup with minimum partition granularity. The user first needs to plan the partition and bucket of the table reasonably, planning partition by time for example, if there is a need for regular data backup. Then carry out regular data backup according to the partition granularity in the later running processes.

Incremental backup can be achieved by backup according to partition granularity.

Data Migration

In order to complete data migration, the user can back up the data to the remote repository first, and then restore the data to another cluster through the remote repository. Because data backup is completed in snapshot form, new imported data after snapshot phase of backup job will not be backed up. Therefore, during the snapshot completion and restoring job completion, the data imported on the original cluster needs to be imported on the new cluster.

It is recommended to concurrently import the new and old clusters for a period of time after the migration. The services will be migrated to a new cluster after the verification of data and service correctness.

Materialized View

Materialized view

Materialized view is a special table that stores the data set that has been calculated in advance (according to the defined SELECT statement) in Palo.

Materialized view is mainly to meet the needs of users, it can not only analyze any dimension of the original detail data, but also analyze and query the fixed dimension quickly.

Applicable scenarios

- Analysis requirements detailed data query and fixed dimension query.
- Queries involve only a small number of columns or rows in a table.
- Query contains some time-consuming processing operations, such as long time aggregation.
- Queries need to match different prefix indexes.

Advantages

- The query performance for those using the same subquery results repeatedly is greatly improved.
- Palo maintains the data of materialized view automatically and ensures the data consistency of base table and materialized view whether they are new loading or deletion operation. No additional labor maintenance costs are required.
- It will automatically match to the optimal materialized view and read the data directly from it during querying.

> Data of automatic maintenance of materialized view will cause some maintenance overhead, which will be explained in the subsequent limitations of materialized view.

Materialized view VS Rollup

Before materialized view function, users generally use Rollup function to improve query efficiency through pre-aggregation. But Rollup has some limitations and it can't do pre-aggregation based on detail model.

Covering Rollup function, materialized view also supports more abundant aggregate functions, which means that materialized view is actually a superset of Rollup.

In other words, the functions supported by the previous ALTER TABLE ADD ROLLUP syntax can now be implemented through CREATE MATERIALIZED VIEW.

Using materialized views

Palo system provides a complete set of DDL syntax for materialized views, including creating, viewing and deletion. The DDL syntax is consistent with PostgreSQL and Oracle.

Create materialized views

First, you need to decide what kind of materialized view to create according to the characteristics of your query statement. That is not to say, however, that it is best that your materialized view definition is identical to one of your queries. And we have two principles:

1. **Abstract** from the query statement, the grouping and aggregation methods shared by multiple queries are defined as materialized views.
2. There is no need to create materialized views for all dimension combinations.

First of all, if a materialized view is abstracted, multiple queries can match it. This is the best materialized view because the maintenance of materialized view itself also needs resource consumption.

If the materialized view only fits a special query, no other query can use it. And it will lead to the low cost performance of this materialized view, which not only takes up the storage resources of the cluster, but also cannot serve more queries.

So the user needs to abstract some materialized view definitions by combining own query statements and data dimension information.

The second is that in the actual analysis query, it will not cover all dimension analysis. Therefore, to create materialized view for common dimension combination can achieve a balance of space and time.

Create materialized views with the following command. Creating materialized view is an asynchronous operation, that is, after the user successfully submits the creation task, Palo calculates the stock data in the background until the view is successfully created.

Refer to CREATE MATERIALIZED VIEW for specific syntax.

Supported aggregate functions

At present, aggregation functions supported by materialized view creation statements are as follows:

- SUM, MIN, MAX
- COUNT, BITMAP_UNION, HLL_UNION
- The form of BITMAP_UNION must be:`BITMAP_UNION(TO_BITMAP(COLUMN))` , column can only be of type integer or `BITMAP_UNION(COLUMN)` （largeint is not supported, either) , and the base table is of AGG model.
- The form of HLL_UNION must be:`HLL_UNION(HLL_HASH(COLUMN))` , column cannot be of type DECIMAL or `HLL_UNION(COLUMN)`, and the base table is of AGG model.

Update strategy

Palo will synchronize the operations of loading and deleting base tables into materialized view table to ensure the consistency of data between materialized view and base table. Also, Palo will improve the update efficiency through incremental update and guarantee the atomicity through transactions.

If the user, for example, inserts data into the base table through INSERT command, the data will be synchronized into the materialized view. Only when the base table and materialized view table are successfully written can INSERT command return successfully.

Query of automatic matching

After the materialized view is created successfully, the user's query does not need any change, that is, the base table of the query. Palo will automatically select an optimal materialized view according to the current query statement, and read data from the materialized view for calculation.

The user can check whether materialized views are used in the current query through EXPLAIN command.

Matching relationship between aggregation in materialized view and aggregation in query:

Aggregation in materialized view	Aggregation in query
sum	sum
min	min
max	max
count	count
bitmap_union	bitmap_union, bitmap_union_count, count(distinct)
hll_union	hll_raw_agg, hll_union_agg, ndv, approx_count_distinct

After the aggregate function of bitmap and hll matches the materialized view, the aggregation operator of the query will be rewritten according to the table structure of the materialized view. Refer to Example 2 for details.

Query of materialized view

Check the kinds of materialized views of the current table and their table structure through the following commands:

```
MySQL [test]> desc mv_test all;
+----------+--------------+---------------+---------+------+-------+---------+-------------+
| IndexName | IndexKeysType | Field         | Type    | Null | Key   | Default | Extra       |
+----------+--------------+---------------+---------+------+-------+---------+-------------+
| mv_test   | DUP_KEYS     | k1            | INT     | Yes  | true  | NULL    |             |
|           |              | k2            | BIGINT  | Yes  | true  | NULL    |             |
|           |              | k3            | LARGEINT | Yes | true  | NULL    |             |
|           |              | k4            | SMALLINT | Yes | false | NULL    | NONE        |
|           |              |               |         |      |       |         |             |
| mv_2      | AGG_KEYS     | k2            | BIGINT  | Yes  | true  | NULL    |             |
|           |              | k4            | SMALLINT | Yes | false | NULL    | MIN         |
|           |              | k1            | INT     | Yes  | false | NULL    | MAX         |
|           |              |               |         |      |       |         |             |
| mv_3      | AGG_KEYS     | k1            | INT     | Yes  | true  | NULL    |             |
|           |              | to_bitmap(`k2`) | BITMAP | No   | false |         | BITMAP_UNION |
|           |              |               |         |      |       |         |             |
| mv_1      | AGG_KEYS     | k4            | SMALLINT | Yes | true  | NULL    |             |
|           |              | k1            | BIGINT  | Yes  | false | NULL    | SUM         |
|           |              | k3            | LARGEINT | Yes | false | NULL    | SUM         |
|           |              | k2            | BIGINT  | Yes  | false | NULL    | MIN         |
+----------+--------------+---------------+---------+------+-------+---------+-------------+
```

As we can see, the current `mv_test` table has three materialized views: mv_1, mv_ 2 and mv_ 3, and their table structures can

also be seen.

Deletion of materialized view

You can delete materialized view through the following command if you no longer need it: CROP MATERIALIZED VIEW.

Best practice 1

Materialized view usage is generally divided into the following steps:

1. Create a materialized view
2. Asynchronously check whether the materialized view is completed
3. Query and automatically match materialized views

First stem: create a materialized view

Suppose the user has a sales record list, storing the transaction id, salesperson, selling store, selling time and amount of each transaction. Create the table with the following statements:

```
create table sales_records(record_id int, seller_id int, store_id int, sale_date date, sale_amt bigint) distributed by hash(record_id) properties("replication_num" = "1");
```

The structure of the table `sales_records` is as follows:

```
MySQL [test]> desc sales_records;
+----------+--------+------+-------+---------+-------+
| Field    | Type   | Null | Key   | Default | Extra |
+----------+--------+------+-------+---------+-------+
| record_id | INT    | Yes  | true  | NULL    |       |
| seller_id | INT    | Yes  | true  | NULL    |       |
| store_id  | INT    | Yes  | true  | NULL    |       |
| sale_date | DATE   | Yes  | false | NULL    | NONE  |
| sale_amt  | BIGINT | Yes  | false | NULL    | NONE  |
+----------+--------+------+-------+---------+-------+
```

At this time, if the user often makes an analysis and query on the sales volume of different stores, then create a materialized view of the sum of sales of the same selling stores grouped by selling stores for the table `sales_records` with the following statements:

```
create materialized view store_amt as select store_id, sum(sale_amt) from sales_records group by store_id;
```

If it returns successfully, the task of creating materialized view is submitted successfully.

Step 2: check whether the materialized view is completed

Because creating materialized view is an asynchronous operation, after submitting the task of creating materialized view, the user needs to check whether the materialized view has been completed asynchronously through the command, which is as follows:

```
SHOW ALTER TABLE MATERIALIZED VIEW FROM db_name;
```

Refer to SHOW ALTER TABLE MATERIALIZED VIEW for more help of this command.

Step 3: query

When the materialized view is created and when the user queries the sales volume of different stores again, the aggregated data in created materialized view `store_amt` will be read directly to improve the query efficiency.

The table `sales_records` is still specified by the user for query, for example:

```
SELECT store_id, sum(sale_amt) FROM sales_records GROUP BY store_id;
```

The above query will automatically match to `store_amt`. The user can check whether the current query matches the appropriate materialized view through the following commands.

```
MySQL [test]> EXPLAIN SELECT store_id, sum(sale_amt) FROM sales_records GROUP BY store_id;
+------------------------------------------------------------------------------+
| Explain String                                                               |
+------------------------------------------------------------------------------+
| PLAN FRAGMENT 0                                                              |
|  OUTPUT EXPRS:<slot 2> `store_id` | <slot 3> sum(`sale_amt`)                 |
|   PARTITION: UNPARTITIONED                                                   |
|                                                                              |
|   RESULT SINK                                                                |
|                                                                              |
|   4:EXCHANGE                                                                 |
|                                                                              |
| PLAN FRAGMENT 1                                                              |
|  OUTPUT EXPRS:                                                               |
|   PARTITION: HASH_PARTITIONED: <slot 2> `store_id`                           |
|                                                                              |
|   STREAM DATA SINK                                                           |
|     EXCHANGE ID: 04                                                          |
|     UNPARTITIONED                                                            |
|                                                                              |
|   3:AGGREGATE (merge finalize)                                               |
|   |  output: sum(<slot 3> sum(`sale_amt`))                                   |
|   |  group by: <slot 2> `store_id`                                           |
|   |                                                                          |
|   2:EXCHANGE                                                                 |
|                                                                              |
| PLAN FRAGMENT 2                                                              |
|  OUTPUT EXPRS:                                                               |
|   PARTITION: RANDOM                                                          |
|                                                                              |
|   STREAM DATA SINK                                                           |
|     EXCHANGE ID: 02                                                          |
|     HASH_PARTITIONED: <slot 2> `store_id`                                    |
|                                                                              |
|   1:AGGREGATE (update serialize)                                             |
|   |  STREAMING                                                               |
|   |  output: sum(`sale_amt`)                                                 |
|   |  group by: `store_id`                                                    |
|   |                                                                          |
|   0:OlapScanNode                                                             |
|      TABLE: sales_records                                                    |
|      PREAGGREGATION: ON                                                      |
|      partitions=1/1                                                          |
|      rollup: store_amt                                                       |
|      tabletRatio=10/10                                                       |
|      tabletList=22038,22040,22042,22044,22046,22048,22050,22052,22054,22056 |
|      cardinality=0                                                           |
|      avgRowSize=0.0                                                          |
|      numNodes=1                                                              |
+------------------------------------------------------------------------------+
45 rows in set (0.006 sec)
```

Among them, the most important one is Rollup property in OlapScanNode. We can see that the Rollup of the current query displays `store_amt`, which means the query has been correctly matched to the materialized view `store_amt`, and can read data directly from materialized view.

Best practice 2 PV,UV

Business scenario: calculate UV and PV of advertisement

Suppose that the user's original ad click data are stored in Palo, for ad PV, UV query can be sped up by creating a materialized view `bitmap_union` .

First, create a table to store the click data details of ads, including the click events of each click, what ads are clicked, through what channels, and who clicked the ads.

```
MySQL [test]> create table advertiser_view_record(time date, advertiser varchar(10), channel varchar(10), user_id int)
distributed by hash(time) properties("replication_num" = "1");
Query O
K, 0 rows affected (0.014 sec)
```

The structure of original ad click data table is as follows:

```
MySQL [test]> desc advertiser_view_record;
+-----------+------------+------+-------+---------+-------+
| Field     | Type       | Null | Key   | Default | Extra |
+-----------+------------+------+-------+---------+-------+
| time      | DATE       | Yes  | true  | NULL    |       |
| advertiser | VARCHAR(10) | Yes  | true  | NULL    |       |
| channel   | VARCHAR(10) | Yes  | false | NULL    | NONE  |
| user_id   | INT        | Yes  | false | NULL    | NONE  |
+-----------+------------+------+-------+---------+-------+
4 rows in set (0.001 sec)
```

1. Create a materialized view

   Since the user wants to query the UV value of the advertisement, which means the user of the same advertisement needs to be accurately de-duplicated, then the query is generally as follows:

```
SELECT advertiser, channel, count(distinct user_id) FROM advertiser_view_record GROUP BY advertiser, channel;
```

For this kind of UV scenario, we can create a materialized view with `bitmap_union` to achieve a pre-accurate de-duplication.

In Palo, the aggregation results of`count (distinct)` and `bitmap_union_count` *are fully identical. And* `bitmap_union_count` *equals* `bitmap_union` *result of count, so if **the query involves* `count(distinct)` *, you can use > to create a materialized view with bitmap* union aggregation to speed up query.**

For this case, you can create a materialized view based on advertisement and channel group for precise de-duplication of `user_id`.

```
MySQL [test]> create materialized view advertiser_uv as select advertiser, channel, bitmap_union(to_bitmap(user_id))
from advertiser_view_record group by advertiser, channel;
Query OK, 0 rows affected (0.012 sec)
```

> Note: Because user_id itself is of INT type, you need to convert the filed to bitmap type through function `to_bitmap` before `bitmap_union` aggregation in Palo.

After the creation, the table structures of ad click list and materialized view are as follows:

```
MySQL [test]> desc advertiser_view_record all;
+----------------------+---------------+---------------------+-----------+-------+-------+---------+-------------+
| IndexName            | IndexKeysType | Field               | Type      | Null  | Key   | Default | Extra       |
+----------------------+---------------+---------------------+-----------+-------+-------+---------+-------------+
| advertiser_view_record | DUP_KEYS    | time                | DATE      | Yes   | true  | NULL    |             |
|                      |               | advertiser          | VARCHAR(10) | Yes | true  | NULL    |             |
|                      |               | channel             | VARCHAR(10) | Yes | false | NULL    | NONE        |
|                      |               | user_id             | INT       | Yes   | false | NULL    | NONE        |
|                      |               |                     |           |       |       |         |             |
| advertiser_uv        | AGG_KEYS      | advertiser          | VARCHAR(10) | Yes | true  | NULL    |             |
|                      |               | channel             | VARCHAR(10) | Yes | true  | NULL    |             |
|                      |               | to_bitmap(`user_id`) | BITMAP    | No    | false |         | BITMAP_UNION |
+----------------------+---------------+---------------------+-----------+-------+-------+---------+-------------+
```

2. Query of automatic matching

   When the materialized view view is created and the advertisement UV is queried, Palo will automatically query data from created materialized view `advertiser_uv` . The original query statements, for example, are as follows:

```
SELECT advertiser, channel, count(distinct user_id) FROM advertiser_view_record GROUP BY advertiser, channel;
```

After the materialized view is selected, the actual query will be converted to:

```
SELECT advertiser, channel, bitmap_union_count(to_bitmap(user_id)) FROM advertiser_uv GROUP BY advertiser,
channel;
```

Check whether Palo matched the materialized view through EXPLAIN command:

```
MySQL [test]> explain SELECT advertiser, channel, count(distinct user_id) FROM  advertiser_view_record GROUP BY
advertiser, channel;
+-----------------------------------------------------------------------------------------------------------------+
| Explain String                                                                                                  |
```

```
| Explain String                                                                     |
+------------------------------------------------------------------------------------+
| PLAN FRAGMENT 0
|  OUTPUT EXPRS:<slot 7> `advertiser` | <slot 8> `channel` | <slot 9>
bitmap_union_count(`default_cluster:test`.`advertiser_view_record`.`mv_bitmap_union_user_id`) |
|   PARTITION: UNPARTITIONED
|
|                                                                                    |
|   RESULT SINK                                                                      |
|                                                                                    |
|   4:EXCHANGE                                                                       |
|                                                                                    |
| PLAN FRAGMENT 1                                                                    |
|  OUTPUT EXPRS:                                                                     |
|   PARTITION: HASH_PARTITIONED: <slot 4> `advertiser`, <slot 5> `channel`
|
|                                                                                    |
|   STREAM DATA SINK
|     EXCHANGE ID: 04                                                                |
|     UNPARTITIONED                                                                  |
|                                                                                    |
|   3:AGGREGATE (merge finalize)
|
|   |  output: bitmap_union_count(<slot 6>
bitmap_union_count(`default_cluster:test`.`advertiser_view_record`.`mv_bitmap_union_user_id`))  |
|   |  group by: <slot 4> `advertiser`, <slot 5> `channel`
|
|   |                                                                                |
|   2:EXCHANGE                                                                       |
|                                                                                    |
| PLAN FRAGMENT 2                                                                    |
|  OUTPUT EXPRS:                                                                     |
|   PARTITION: RANDOM
|                                                                                    |
|   STREAM DATA SINK
|     EXCHANGE ID: 02                                                                |
|     HASH_PARTITIONED: <slot 4> `advertiser`, <slot 5> `channel`
|
|                                                                                    |
|                                                                                    |
|   1:AGGREGATE (update serialize)
|
|   |  STREAMING                                                                     |
|   |  output: bitmap_union_count(`default_cluster:test`.`advertiser_view_record`.`mv_bitmap_union_user_id`)
|
|   |  group by: `advertiser`, `channel`
|   |                                                                                |
|   0:OlapScanNode                                                                   |
|      TABLE: advertiser_view_record
|
|      PREAGGREGATION: ON
|      partitions=1/1                                                                |
|      rollup: advertiser_uv                                                         |
|      tabletRatio=10/10                                                             |
|      tabletList=22084,22086,22088,22090,22092,22094,22096,22098,22100,22102
|
|      cardinality=0                                                                 |
|      avgRowSize=0.0                                                                |
|      numNodes=1                                                                    |
+------------------------------------------------------------------------------------+
45 rows in set (0.030 sec)
```

In EXPLAIN results, we can first see that the value of Rollup property of OlapScanNode is advertiser_ uv which means the query directly scans the materialized view data, indicating that the match is successful.

Second, for the field user*id`, count (distinct) is rewritten to bitmap* union *count(to* bitmap)`to achieve accurate de-duplication through bitmap.

Best practice 3

Business scenario: Matching richer prefix index

The user's original table has three columns （k1, k2, k3）, where k1 and k2 are prefix index columns. In this case, if the user's query criteria contain `where k1=1 and k2=2`, the query can be accelerated through the index.

However, in some cases, the user's filtering conditions cannot match the prefix index, `where k3=3` for example. Then the query speed cannot be improved by index.

Create a materialized view with k3 as the first column to solve this problem.

1. Create a materialized view

```
CREATE MATERIALIZED VIEW mv_1 as SELECT k3, k2, k1 FROM tableA ORDER BY k3;
```

After creating the view through the above syntax, the materialized view retains the complete detail data, and its prefix index is column k3. Here is the table structure:

```
MySQL [test]> desc tableA all;
+----------+--------------+-------+-----+------+-------+---------+-------+
| IndexName | IndexKeysType | Field | Type | Null | Key   | Default | Extra |
+----------+--------------+-------+-----+------+-------+---------+-------+
| tableA    | DUP_KEYS     | k1    | INT | Yes  | true  | NULL    |       |
|           |              | k2    | INT | Yes  | true  | NULL    |       |
|           |              | k3    | INT | Yes  | true  | NULL    |       |
|           |              |       |     |      |       |         |       |
| mv_1      | DUP_KEYS     | k3    | INT | Yes  | true  | NULL    |       |
|           |              | k2    | INT | Yes  | false | NULL    | NONE  |
|           |              | k1    | INT | Yes  | false | NULL    | NONE  |
+----------+--------------+-------+-----+------+-------+---------+-------+
```

2. Query matching

At this time, if the filter condition is that there is column K3 in the user's query, for example:

```
select k1, k2, k3 from table A where k3=3;
```

then the query will read data directly from the created materialized view mv_1 . Materialized view has prefix index to k3, so the query efficiency will be improved.

Limitations

1. The parameter of aggregate function of materialized view supports only single column rather than expressions, sum (a+b), for example, is not supported.
2. If the condition column of the deletion statement is not in the materialized view, the deletion operation cannot be performed. Delete materialized view first before deleting data if you really need to delete data.
3. Too many materialized views on a single table will affect the loading efficiency: materialized views and base table data are updated synchronously when loading data, if there are more than 10 materialized views in a table, the loading speed may be very slow, which is similar to loading data of 10 tables simultaneously for a single load.
4. The same column and different aggregation functions cannot appear in one materialized view simultaneously, select sum(a), min(a) from table are not supported, for example.

Exception error

1. DATA_QUALITY_ERR: "The data quality does not satisfy, please check your data"

   This is caused by failed materialized view creation due to data quality problems. Note: bitmap type only supports positive integers, if there are negative integers in original data, the creation of materialized views will fail.

Privilege Management

Privilege management system of Palo refers to the privilege management mechanism of Mysql achieving table level fine-grained permission control and supporting white list mechanism.

Word explanation

1. User identity `user_identity`

   In the privilege system, a user is identified by a `User Identity`. User identity consists of two parts: `user name and userhost`. `username` is defined as the user name, which is composed of English cases. `userhost` indicates the IP address of the user links. `user_identity`is presented in the form of`username@'userhost'` , indicating the `username` from `userhost`.

   Another presentation of`user_identity` is `username@['domain']`, where `domain` means domain name, which can be resolved to a group of ip through DNS or BNS (Baidu name service). The final presentation is a group of `username@'userhost'`, so we use `username@'userhost'` to present in the following.

2. Privilege `Privilege`

   The objects of privilege are nodes, databases, tables or resources. Different privileges represent different operation privileges.

3. Role `Role`

   Palo can create custom-named roles which can be seen as a collection of a set of privileges. A newly-created user can be

given a role, which means the user will be automatically given the privileges owned by the role. Subsequent privilege changes to a role will also be reflected in the all privileges of the user belonging to this role.

4. User property `user_property`

   User property is directly attached to a user, not a user ID. That is, both `cmy@'192.%'` and `cmy@['domain']`have the same set of user properties, which belong to the user `cmy` , not`cmy@'192.%'`or`cmy@['domain']`.

   User properties include but are not limited to: maximum number of user connections, load cluster configuration, etc.

Supported operations

1. Create a user: CREATE USER
2. Delete a user: DROP USER
3. Grant: GRANT
4. Revoke: REVOKE
5. Create a role: CREATE ROLE
6. Drop a role: DROP ROLE
7. View user privileges: SHOW GRANTS
8. View created roles: SHOW ROELS
9. View use property: SHOW PROPERTY
10. Set user property: SET PROPERTY

Privilege descriptions

Palo currently supports the following privileges:

1. `Node_priv`

   It means node changing privilege, including addition, deletion, offline and other operations of FE, BE, BROKER nodes. At present, this privilege can only be granted to Root users.

2. `Grant_priv`

   It means changing privilege, which allows the performances such as granting, revocation, addition / deletion / change of user / role, etc.

3. `Select_priv`

   It means privilege of read-only for databases and tables.

4. `Load_priv`

   It means privilege of writing for databases and tables, including Load、Insert、Delete and other operations.

5. `Alter_priv`

   It means privilege of changing databases and tables, including renaming database / table, addition / deletion / changing column and other operations.

6. `Create_priv`

   It means privilege of creating databases and tables.

7. `Drop_priv`

   It means privilege of deleting databases and tables.

8. `Usage_priv`

   it means privilege of using some Resources.

Best practices

1. The following users and roles will be created automatically when Palo is initialized:
   1. Role admin: The role has Admin_priv, which means all privileges except node change.
   2. admin@'%' : the user admin, the user is allowed to log in from any node, and the role is admin.
2. Deleting or changing privileges for roles or users created by default is not supported.
3. Users of the role admin can be created more than one.
4. Some instructions that may lead to conflicts
   1. Domain name and ip conflict:

      Suppose the following user is created:

```
CREATE USER cmy@['domain'];
```

      and be granted with authorization:

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

      The domain name is resolved into two ip: iP1 and iP2

      After supposing, we grant a separate authorization to `cmy@'ip1'`:

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

Then the privilege of `cmy@'ip1'` is changed to `SELECT_PRIV and ALTER_PRIV`.And when we change the privilege of `cmy@['domain']` again, `cmy@'ip1'`will not follow the change.

2. Repeated ip conflict:

   Suppose the following user is created:

```
CREATE USER cmy@'%' IDENTIFIED BY "12345";
CREATE USER cmy@'192.%' IDENTIFIED BY "abcde";
```

   In terms of priority,`'192.%'` takes precedence over`'%'`. Therefore, when the user `cmy` attempts to log in to Palo with password `'12345'`from computer `192.168.1.1`, the user will be rejected.

5. When the user forgets the password

   Public cloud user can refer to Reset administrator password.

   Refer to SET PASSWORD command to reset the password after login.

Resource Management

Similar to a property collection, Resource is mainly used to define some common properties for unified management or reference in other places.

Currently, Palo supports the following resources:

- ODBC Resource

  Used to define a set of information that connects to an external data source through ODBC and can be used in Create ODBC external table.

Basic concepts

A resource contains name, type and other basic information and the name is globally unique, different types of resources contain different properties, please refer to the introductions of each resource for details.

The creation and deletion of resources can only be performed by users with `admin` privilege.

Users with `admin` privilege can give the use privilege `usage_priv` to ordinary users. Refer to Privilege Management for details.

Specific operations

There are mainly three commands in resource management

1. CREATE RESOURCE : create a resource.
2. DROP RESOURCE : delete a resource.
3. [ALTER RESOURCE] : please wait and see.
4. SHOW RESOURCES : present created resources.

Best practice

1. Manage ODBC connection information.

   The user may create multiple ODBC external tables to map multiple tables of external data sources. If the connection information of the external data source changes, such as the user names, passwords, and connection address, the mapping table needs to be re created in Palo, which is rather inconvenient.

   While a group of connection information can be saved in a resource in the form of resources for the references of external tables when they are created. When the connection information changes, you just need to directly modify the properties in this resource.

   > Resource modification function is coming soon. If you need to modify currently, you need to delete the resource and re-create the resource with the same name.

Variable

The Variable in Palo refer to the variable settings in MySQL.

Users can set session level variables and globally effective scalars.

Some variables, however, are only used to be compatible with some MySQL client protocols, and do not have their practical significance in MySQL database.

Variable Setting and Viewing

Variable viewing

```
SHOW VARIABLES;
SHOW VARIABLES LIKE '%time_zone%';
```

Variable setting

Some variables can be set to be effective globally or only for the current session. After the setting of being effective globally, the setting value will be used in subsequent new session connections. The setting is only effective for the current session, and the variable will only affect the current session.

The setting of being effective for only the current session:

```
 SET exec_mem_limit = 137438953472;
SET forward_to_master = true;
SET time_zone = "Asia/Shanghai";
```

The setting of being effective globally:

```
SET GLOBAL exec_mem_limit = 137438953472
```

> Note 1： Only the user admin can set globally effective variables. Note 2： Globally effective variables only affect the variables in the new session rather than the variable values of the current session. The setting is effective after disconnecting and logging in again.

Variables that support both the current session and global effectiveness include:

- `time_zone`
- `wait_timeout`
- `sql_mode`
- `is_report_success`
- `query_timeout`
- `exec_mem_limit`
- `batch_size`
- `parallel_fragment_exec_instance_num`
- `parallel_exchange_instance_num`
- `enable_fold_constant_by_be`
- `enable_cost_based_join_reorder`

Variables that support only global effectiveness include:

- `default_rowset_type`

At the same time, constant expressions are supported for variable settings. For example:

```
SET exec_mem_limit = 10 * 1024 * 1024 * 1024;
SET forward_to_master = concat('tr', 'u', 'e');
```

Set Variables in Query Statements

We, in some scenarios, may need to set variables targeting certain queries. Session variables can be set in the query through SET_VAR prompt（effective within a single statement）. Examples are as follows:

```
SELECT /*+ SET_VAR(exec_mem_limit = 8589934592) */ name FROM people ORDER BY name;
SELECT /*+ SET_VAR(query_timeout = 1) */ sleep(3);
```

Note: Notes must start with `/*+` and only must come after SELECT.

Supported Variables

- `SQL_AUTO_IS_NULL`

  Used for compatibility with JDBC connection pool C3P0. No practical effect.

- `auto_increment_increment`

  Used for compatibility with MySQL client. No practical effect.

- `autocommit`

  Used for compatibility with MySQL client. No practical effect.

- `batch_size`

  Used to specify the number of rows of a single data packet transmitted by each nodes during query execution. The number of rows in a data packet, by default, is 1024, which means that every 1024 rows of data generated by the source node will be packaged and sent to destination node.

  A larger number of rows will enhance the query throughput in the scenario of scanning large data amount, but may increase the query delay in the scenario of small query. At the same time, it also increases the memory cost of the query. The recommended setting range is 1024 to 4096.

- `character_set_client`

  Used for compatibility with MySQL client. No practical effect.

- `character_set_connection`

  Used for compatibility with MySQL client. No practical effect.

- character_set_results

  Used for compatibility with MySQL client. No practical effect.

- character_set_server

  Used for compatibility with MySQL client. No practical effect.

- codegen_level

  Used to set LLVM codegen level（Currently not effective）.

- collation_connection

  Used for compatibility with MySQL client. No practical effect.

- collation_database

  Used for compatibility with MySQL client. No practical effect.

- collation_server

  Used for compatibility with MySQL client. No practical effect.

- disable_colocate_join

  Used to control whether to start Colocation Join function. The default value is false, indicating that the function is started. While true means that the function is prohibited. When this function is prohibited, query plan will not attempt to execute Colocation Join.

- disable_streaming_preaggregations

  Used to control whether to start streaming preaggregations. The default value is false, indicating that the function is started. It cannot be set currently and is started by default.

- enable_cost_based_join_reorder

  Whether to start Join Reorder optimization based on cost model. It is off by default. After starting, Palo will use a better planning method to determine the order of Join. The function is still experimental, though, for complex table join queries, the user can try to start the function to observe the optimization effect.

- enable_fold_constant_by_be

  Whether to evaluate constant expressions in SQL through ComputeNode. The default is false.

  The Leader Node will calculate some constant expressions in SQL by default, which is helpful for query planner to perform query optimizations such as predicate push down and partition clipping for the calculation of constant expressions. The calculation ability of Leader Node, however, is limited, which can not support some complex constant expressions. At this time, the user can set this variable to true and Palo will use Compute Node to calculate the expression. However, this method will increase additional RPC between nodes. The performance impact on the cluster should be observed for high concurrency scenarios.

- enable_insert_strict

  Used to set whether `strict`mode is started when importing data through INSERT statement. The default is false, which means `strict` mode is off. Refer to Here for mode instructions.

- enable_spilling

  Used to set whether to start large data volume drop sorting. The default is false, which means the function is off. When the user does not specify the LIMIT condition of the ORDER BY clause, the drop sorting can be started with `enable_spilling` set to true simultaneously. When the function is started, temporary drop data will be stored in `doris-scratch/` directory in BE data directory, and the temporary data will be cleared after the query is finished.

  This function is mainly used to sort large amount of data with limited memory.

  Note that this function is experimental and its stability can not be guaranteed. Please start it carefully.

- exec_mem_limit

  Used to set the memory limit for a single query. The default is 2GB, the unit is B/K/KB/M/MB/G/GB/T/TB/P/PB, and the default is B.

  This parameter is used to limit the memory used in a single query plan instance in a query plan. A query plan may have multiple instances, and a BE node may execute one or more instances. Therefore, this parameter can not accurately limit the memory usage of a query in the whole cluster, nor can it accurately limit the memory usage of a query in a single BE node. The details need to be determined according to the generated query plan.

  Often, only some blocking nodes (such as sorting node, aggregation node and Join node) will consume more memory, while other nodes (such as scan node) will not consume more memory because the data is streaming.

  When Memory Exceed Limit error occurs, the user can try to increase the parameter exponentially, such as 4G, 8G, 16g, etc.

  forward_to_master

  User to set whether to forward some commands to Master FE node for execution. The default is false, which means no forwarding. Palo has several FE nodes, one of which is the Master node. Generally, the user can connect any FE node for full function operation. However, only the Master FE node can obtain the detailed information for some information viewing instructions.

  SHOW BACKENDS; command, for example, if the user does not forward to the Master FE node, the user can only see some basic information, such as whether the node is alive or not, while if the user forward to the Master FE node, the user can get more detailed information, including the start time of the node, the last heartbeat time and so on.

  Commands currently affected by this parameter are as follows:

  1. SHOW FRONTENDS;

     Forward to Master to view the last heartbeat information.

  2. SHOW BACKENDS;

     Forward to Master to view the start time, last heartbeat information and disk capacity information.

  3. SHOW BROKER;

     Forward to Master to view the start time, last heartbeat information

  4. SHOW TABLET;/ADMIN SHOW REPLICA DISTRIBUTION;/ADMIN SHOW REPLICA STATUS;

     Forward to Master to view the tablet information stored in Master FE metadata. Normally, the tablet information in different FE metadata should be consistent. Compare the differences between the current FE and Master FE metadata with this method when problems occur.

  5. SHOW PROC;

     Forward to Master to view the relevant PROC information stored in the Master FE metadata. It is mainly used for metadata comparison.

- init_connect

  Used for compatibility with MySQL client. No practical effect.

- interactive_timeout

  Used for compatibility with MySQL client. No practical effect.

- is_report_success

  Used to set whether to view the profile of query. The default is false, which means no profile is required.

  By default, BE will send profile to FE to view the error only when there is an error in the query. Normally ended queries do not send profile. Sending profile will cause some network overhead, which is adverse for high concurrent query scenarios. When the user wants to analyze the profile of a query, the user can set this variable to true and send the query. After the

query, the user can view the profile on the web page of the currently connected FE:

`fe_host:fe_http_port/query`

The latest 100 items of profile queried when `is_report_success` is started will be displayed.

- `language`

  Used for compatibility with MySQL client. No practical effect.

- `license`

  Used to display the License of Palo. No other effects.

- `load_mem_limit`

  Used to specify the memory limit for the load operation. The default value is 0, which means that the variable is not used as the memory limit of the loading operation, but `exec_mem_limit` is used.

  This variable is only used for INSERT operations. Because INSERT operation is designed for 2 parts: query and loading, and if the user does not set this variable, the memory limit of the query and loading operation is`exec_mem_limit` respectively. Otherwise, the memory limit of INSERT query is `exec_mem_limit`, and the loading limit is `load_mem_limit`.

  Memory limits of other loading methods such as BROKER LOAD and STREAM LOAD still use `exec_mem_limit`.

- `lower_case_table_names`

  Used for compatibility with MySQL client. Cannot be set. The current table names in Palo are case sensitive by default.

- `max_allowed_packet`

  Used for compatibility with JDBC connection pool C3P0. No practical effect.

- `net_buffer_length`

  Used for compatibility with MySQL client. No practical effect.

- `net_read_timeout`

  Used for compatibility with MySQL client. No practical effect.

- `net_write_timeout`

  Used for compatibility with MySQL client. No practical effect.

- `parallel_exchange_instance_num`

  Used to set the number of exchange nodes used by an upper node to receive data from a lower node in the execution plan. The default value is - 1, which means that the number of exchange nodes is equal to the number of execution instances of the lower level nodes (the default behavior). When the setting is greater than 0 and less than the number of execution instances of lower level nodes, the number of exchange nodes is equal to the set value.

  In a distributed query execution plan, the upper node usually has one or more exchange nodes to receive data from the execution instances of the lower level nodes on different BE. Usually, the number of exchange nodes is equal to the number of execution instances of the lower level nodes.

  In some aggregation query scenarios, on DUPLICATE KEY detail model, for example, if the amount of data that needs to be scanned at the bottom is large, but the amount of data after aggregation is small, the user can try to modify this variable to a smaller value to reduce the resource cost of such queries.

- `parallel_fragment_exec_instance_num`

  For Scan nodes, Used to set the number of execution instances on each BE node for Scan nodes. The default is 1.

  Usually, a query plan generates a set of scan ranges, which are the data ranges to be scanned. These data are distributed on multiple BE nodes and a BE node will have one or more scan ranges. A set of scan ranges for each BE node, by default, is handled by only one execution instance. To improve the query efficiency, this variable can be added, when the machine resources are abundant, to allow more execution instances to process a set of scan ranges simultaneously.

The number of scan instances determines the number of other execution nodes in the upper layer, such as aggregation nodes and join nodes. It is, therefore, equivalent to increasing the concurrency of the whole query plan execution. Modifying this parameter will improve the efficiency of large query, but larger values will consume more machine resources, such as CPU, memory, disk IO.

- `query_cache_size`

  Used for compatibility with MySQL client. No practical effect.

- `query_cache_type`

  Used for compatibility with JDBC connection pool C3P0. No practical effect.

- `query_timeout`

  Used to set the query timeout. This variable will act on all the query statements like INSERT statement in the current connection. The default is 5 minutes in seconds.

- `resource_group`

  Out of use for the time being.

- `sql_mode`

  Used to specify the SQL mode to accommodate certain SQL dialects. Refer to Here for SQL mode.

- `sql_safe_updates`

  Used for compatibility with MySQL client. No practical effect.

- `sql_select_limit`

  Used for compatibility with MySQL client. No practical effect.

- `system_time_zone`

  Use to display the current system time zone. It is unalterable.

- `time_zone`

  Used to set the time zone, which affects the results of some time functions, for the current session. Refer to Here for time zone.

- `tx_isolation`

  Used for compatibility with MySQL client. No practical effect.

- `version`

  Used for compatibility with MySQL client. No practical effect.

- `performance_schema`

  Used for compatibility with MySQL JDBC with 8.0.16 and above. No practical effect.

- `version_comment`

  Used to display Palo version. It is unalterable.

- `wait_timeout`

  Used to set the connection duration of idle connections. Palo will initiatively disconnect the connection when an idle connection has no interaction with Palo during the period. The default is 8 hours in seconds.

- `rewrite_count_distinct_to_bitmap_hll`

  Used to decide whether to rewrite count distinct queries of type bitmap and hll as bitmap_union_count and hll_union_agg.

- `prefer_join_method`

Used to decide that when selecting the specific implementation mode of a join as broadcast join or shuffle join and if broadcast join cost and shuffle join cost are equal, which join mode is preferred.

The currently optional values of this variable are `broadcast` or `shuffle`.

Time Zone

Palo supports multi time zone settings

There are multiple Variables related with time zone in Palo.

- `system_time_zone` : When the server is started, it will be automatically set according to the time zone set by the machine, which cannot be altered after setting.
- `time_zone` : The current time zone of the server includes session level time zone and global level time zone.

Specific operations

1. `show variables like '%time_zone%'`

   Used to view the current time zone configuration.

2. `SET time_zone = 'Asia/Shanghai'`

   Used to set session level time zone, which will be invalid when the connection is disconnected.

3. `SET global time_zone = 'Asia/Shanghai'`

   Used to set global level time zone parameter, which will not be invalid when the connection is disconnected.

The effects of time zone

Time zone setting will affect the display and storage of time value sensitive to time zone,

Including the values displayed by time functions such as NOW or CURTIME and the time values in SHOW LOAD and SHOW BACKENDS.

But it will affect neither the values of time type partition column in CREATE TABLE, nor the display of values stored as `date/datetime` type.

Functions affected by time zone:

- FROM_UNIXTIME: Used to return the date and time of the specified time zone through a given UTC timestamp like `FROM_UNIXTIME(0)`, returning to CST time zone: ``1970-01-01 08:00:00`.
- UNIX_TIMESTAMP: Used to return UTC timestamp through a given date and time of a specified time zone, such as CST time zone `UNIX_TIMESTAMP('1970-01-01 08:00:00')`, returning to 0,
- CURTIME: Used to return the time in the specified time zone.
- NOW: Used to return the date and time in the specified time zone.
- CONVERT_TZ: Used to transfer a date and time from one specified time zone to another.

Restrictions on use

Time zone values can be presented in several formats and are not case sensitive:

- Strings representing UTC offset, such as' + 10:00 'or' - 6:00 '
- Standard time zone format, such as "Asia / Shanghai", "America / Los"_ Angeles"
- Abbreviated time zone formats such as "MET", "CTT" are not supported and not recommended because they are ambiguous in different scenarios.

- In order to be compatible with Palo of old version of Palo and support CST abbreviated time zone, CST will be internally converted standard Chinese time zone like "Asia/Shanghai".

Time zone format list

Refer to List of tz database time zones to get writing ways of all time zones.

Experimental Functions

This document mainly introduces some experimental functions of Palo.

These experimental functions are turned off by default, and will be turned on by default in the later version upgrade after the function matures.

Join Reorder

In the field of data analysis, SQL query optimizer can significantly improve the execution efficiency of SQL submitted by users. One of the most important parts is to automatically adjust the Join order of tables in SQL. Different Join orders might lead to a huge difference in the efficiency of SQL execution.

The user can, in the new version, enable new Join Reorder algorithm logic through the following Session variable. The algorithm will try to avoid Cross Join and optimize the Join order according to the cost model.

```
set enable_cost_based_join_reorder = true;
```

In performing complex join operations, the user can try to enable this function to see if it can improve the execution efficiency.

Constant expression collapse

Palo query optimizer collapses some constant expressions in SQL (directly calculate the constant value and replace the constant expression in the original SQL), examples are as follows:

```
select * from tbl where k1 = 1+2;
==>
select * from tbl where k1 = 3;
```

Due to the limitation of calculation ability of Compute Node (FE), however, some complex constant expressions cannot be calculated, while the unfolded constant expressions may affect the optimization functions such as partition clipping and predicate push down.

In new version, the user can use the full expression calculation ability of Compute Node (BE) to calculate constant expressions by enable the following Session variable:

```
set enable_fold_constant_by_be = true;
```

Note that enabling this function will cause additional RPC overhead. In high concurrency query scenarios, the effect on the whole cluster should be observed.

Operation Guide

Cluster Scaling

After the cluster is created, you can modify the scale of the cluster as needed at any time, and click the `Cluster Scaling` to enter into the cluster scaling page.

The page displays the configuration and billing information of the current user cluster, such as:

You can modify the expected number of clusters. At present, Palo only supports expanding nodes, the feature of contracting nodes is under development.

In which, Leader Node can be expanded to 3, Compute Node can be expanded to at most 50.

> Notes: the nodes in the red box below are the final number of nodes in total, not the number of nodes newly added.

If you are a prepaid user, the configuration fee is for the configuration newly added. If you are a pay-for-use user, the configuration fee is the total unit price cost after capacity expansion.

After operation, click `Cluster Scaling`, confirm and pay the order, then the modification of cluster scale is completed.

The cluster now is entering the status `Expanding`.

At the same time, you can see that the expansion node is being created.

The expansion work can be completed in about 1-5 minutes. And the expansion node will change to the status `Running`.

SQL Manual

Built-in Functions

String Function

Palo supports the following string functions:

1.ascii
2.concat
3.concat_ws

4.ends_with
5.find_in_set
6.group_concat
7.instr
8.length,char_length,character_length
9.locate
10.lower,lcase
11.lpad
12.ltrim
13.money_format
14.null_or_empty
15.parse_url
16.regexp_extract
17.regexp_replace
18.repeat
19.replace
20.reverse
21.rpad
22.rtrim
23.space
24.split_part
25.starts_with
26.strleft,left
27.strright,right
28.substr,substring
29.trim
30.upper,ucase

ASCII

Description

```
ascii(string str)
```

- Function: return the ascii code corresponding to the first string in strings
- Return type: int type

Example

```
mysql> select ascii('palo');
+--------------+
| ascii('palo') |
+--------------+
|          112 |
+--------------+

mysql> select ascii('palo and doris');
+------------------------+
| ascii('palo and doris') |
+------------------------+
|                    112 |
+------------------------+
```

Keywords

```
ascii
```

CONCAT

Description

```
concat(string a, string b...)
```

- Function: connect multiple strings together
- Return type: string type
- Instructions for use: concat () and concat_ws () combine multiple columns in a row into a new column, group_concat () is an aggregate function that combines the results of different rows into a new column

Example

```
mysql> select concat('The date today is ',to_date(now()));
+-------------------------------------------+
| concat('The date today is ', to_date(now())) |
+-------------------------------------------+
| The date today is 2020-12-29              |
+-------------------------------------------+
```

Keywords

```
concat
```

CONCAT_WS

Description

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

- Function: connect the second parameter with parameters behind, the first parameter is the connector.
- Return type: string type

Example

```
mysql> select concat_ws('a', 'b', 'c', 'd');
+-----------------------------+
| concat_ws('a', 'b', 'c', 'd') |
+-----------------------------+
| bacad                       |
+-----------------------------+
```

Keywords

```
concat_ws
```

ENDS_WITH

Description

```
ends_with(string str, string strEnd)
```

- Function: Judge if str ends with strEnd
- Return type: bool type

Example

```
mysql> select ends_with('today','y');
+-----------------------+
| ends_with('today', 'y') |
+-----------------------+
|                     1 |
+-----------------------+
```

Keywords

```
ends_with
```

FIND_IN_SET

Description

```
find_in_set(string str, string strList)
```

- Function: return the position of the first str in strlist (counting from 1). StrList separates multiple strings with commas. If no str is found in strList, then return 0.
- Return type: int type

Example

```
mysql> select find_in_set("beijing", "tianji,beijing,shanghai");
+-----------------------------------------------+
| find_in_set('beijing', 'tianji,beijing,shanghai') |
+-----------------------------------------------+
|                                             2 |
+-----------------------------------------------+
```

Keywords

```
find_in_set
```

GROUP_CONCAT

Description

```
group_concat(string s [, string sep])
```

- Function: the function is an aggregate function similar to sum (), in which, group_concat connects multiple rows of results in the result set into a string. The second parameter is the connector between strings, which can be omitted. The function is often used in combination with the group by statement.

- Return type: string type

Example

```
mysql> select k1, group_concat(k2) from tbl group by k1;
+---+-----------------+
| k1 | group_concat(k2) |
+---------------------+
| 1  | 1,2,3,4          |
+---------------------+
| 1  | 5,6,7,8          |
+---------------------+
```

Keywords

```
group_concat
```

INSTR

Description

```
instr(string str, string substr)
```

- Function: return the location where the substr appears in str for the first time (counting from 1). If no substr appears in str, then return 0.
- Return type: int type

Example

```
mysql> select instr('foo bar bletch', 'b');
+-----------------------------+
| instr('foo bar bletch', 'b') |
+-----------------------------+
|                           5 |
+-----------------------------+

mysql> select instr('foo bar bletch', 'z');
+-----------------------------+
| instr('foo bar bletch', 'z') |
+-----------------------------+
|                           0 |
+-----------------------------+
```

Keywords

```
instr
```

LENGTH

Description

```
length(string a)

char_length(string a)

character_length(string a)
```

- Function: return the length of string. In which, return the length of byte in `length` , and return the length of characters in `char(acter)_length` .
- Return type: int type

Example

```
mysql> select length('today');
+----------------+
| length('today') |
+----------------+
|              5 |
+----------------+

mysql> select length("China");
+----------------+
| length('China')  |
+----------------+
|              6 |
+----------------+

mysql> select char_length("China");
+---------------------+
| char_length('China')  |
+---------------------+
|                   2 |
+---------------------+
```

> Notes: UTF-8 encoding, one Chinese character occupies 3 bytes.

Keywords

```
length, char_length, character_length
```

LOCATE

Description

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

- Function: Return the location where the substr appears in str (counting from 1). If the third parameter is specified, locate the substr in str string from where the string is subscripted with pos.
- Return type: int type

Example

```
mysql> select locate('bj', 'where is bj', 10);
+--------------------------------+
| locate('bj', 'where is bj', 10) |
+--------------------------------+
|                             10 |
+--------------------------------+

mysql> select locate('bj', 'where is bj', 11);
+--------------------------------+
| locate('bj', 'where is bj', 11) |
+--------------------------------+
|                              0 |
+--------------------------------+
```

Keywords

```
locate
```

LOWER,LCASE

Description

```
lower(string a)

lcase(string a)
```

- Function: convert all strings in the parameter to lowercase
- Return type: string type

Example

```
mysql> select lower('toDAY Is FridAy');
+-------------------------+
| lower('toDAY Is FridAy') |
+-------------------------+
| today is friday         |
+-------------------------+

mysql> select lcase('toDAY Is FridAy');
+-------------------------+
| lcase('toDAY Is FridAy') |
+-------------------------+
| today is friday         |
+-------------------------+
```

Keywords

```
lower,lcase
```

LPAD

Description

```
lpad(string str, int len, string pad)
```

- Function: Return a string of length len (starting from the first letter) in str. If the len is greater than the length of str, then add pad characters before str until the length of the string reaches len. If len is smaller than the length of str, this function is equivalent to truncating str string and will return only the string with length len.
- Return type: string type

Example

```
mysql> select lpad('aoaoaoao',10,'xy');
+---------------------------+
| lpad('aoaoaoao', 10, 'xy') |
+---------------------------+
| xyaoaoaoao                |
+---------------------------+

mysql> select lpad('aoaoaoao',6,'xy');
+---------------------------+
| lpad('aoaoaoao', 6, 'xy') |
+---------------------------+
| aoaoao                    |
+---------------------------+
```

Keywords

```
lpad
```

LTRIM

Description

```
ltrim(string a)
```

- Function: remove the spaces that appear continuously from the beginning of the parameter.
- Return type: string type

Example

```
mysql> select ltrim('    today is friday');
+----------------------------+
| ltrim('    today is friday') |
+----------------------------+
| today is friday            |
+----------------------------+
```

Keywords

```
ltrim
```

MONEY_FORMAT

Description

```
money_format(numric money)
```

- Function: convert to money format
- Return type: string type

Example

```
select money_format(11111);
+--------------------+
| money_format(11111) |
+--------------------+
| 11,111.00          |
+--------------------+
```

Keywords

```
money_format
```

NULL_OR_EMPTY

Description

```
null_or_empty(string str)
```

- Function: judge whether the str is NULL or empty string
- Return type: bool type

Example

```
mysql> select null_or_empty('');
+-------------------+
| null_or_empty('') |
+-------------------+
|                 1 |
+-------------------+

mysql> select null_or_empty('today');
+-------------------------+
| null_or_empty('today') |
+-------------------------+
|                       0 |
+-------------------------+
```

Keywords

```
null_or_empty
```

PARSE_URL

Description

```
parse_url(string url, string name)
```

- Function: parse the field corresponding to name in url, there are following options of name: 'PROTOCOL', 'HOST', 'PATH', 'REF', 'AUTHORITY', 'FILE', 'USERINFO', 'PORT', 'QUERY', and then return the result.
- Return type: string type

Example

```
mysql> select parse_url ('https://cloud.baidu.com/product/palo.html', 'PROTOCOL');
+-----------------------------------------------------------------+
| parse_url('https://cloud.baidu.com/product/palo.html', 'PROTOCOL') |
+-----------------------------------------------------------------+
| https                                                           |
+-----------------------------------------------------------------+
1 row in set (0.02 sec)
```

Keywords

```
parse_url
```

REGEXP_EXTRACT

Description

```
regexp_extract(string subject, string pattern, int index)
```

- Function: regular matching the string. Return the entire string matched if the index is 0, return the first, second, ...th part accordingly when the index is 1, 2, ...

- Return type: string type

Example

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

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

Keywords

```
regexp_extract, regexp
```

REGEXP_REPLACE

Description

```
regexp_replace(string initial, string pattern, string replacement)
```

- Function: replace the part in the initial string that matches the pattern with replacement.
- Return type: string type

Example

```
mysql> select regexp_replace('aaabbbaaa','b+','xyz');
+----------------------------------------+
| regexp_replace('aaabbbaaa', 'b+', 'xyz') |
+----------------------------------------+
| aaaxyzaaa                              |
+----------------------------------------+

mysql> select regexp_replace('aaabbbaaa','(b+)','<\\1>');
+----------------------------------------+
| regexp_replace('aaabbbaaa', '(b+)', '<\1>') |
+----------------------------------------+
| aaa<bbb>aaa                            |
+----------------------------------------+

mysql> select regexp_replace('123-456-789','[^[:digit:]]','');
+----------------------------------------------+
| regexp_replace('123-456-789', '[^[:digit:]]', '') |
+----------------------------------------------+
| 123456789                                    |
+----------------------------------------------+
```

Keywords

```
regexp_replace, regexp
```

REPEAT

Description

```
repeat(string str, int n)
```

- Function: return the result after repeating string str n times
- Return type: string type

Example

```
mysql> select repeat("abc", 3);
+----------------+
| repeat('abc', 3) |
+----------------+
| abcabcabc      |
+----------------+
```

Keywords

```
repeat
```

REPLACE

Description

```
replace(string oriStr, string src, string dest)
```

- Function: replace all srcs in oriStr with dest, take the result as the return value. Note the difference between regexp_replace (), replace() is the string matched exactly, and regexp_replace () can support expressions.
- Return type: string type

Example

```
mysql> select replace('aaabbbaaa','b+','xyz');
+----------------------------------+
| replace('aaabbbaaa', 'b+', 'xyz') |
+----------------------------------+
| aaabbbaaa                        |
+----------------------------------+

mysql> select replace('aaabbbaaa','bb','xyz');
+----------------------------------+
| replace('aaabbbaaa', 'bb', 'xyz') |
+----------------------------------+
| aaaxyzbaaa                       |
+----------------------------------+
```

Keywords

```
replace
```

REVERSE

Description

```
reverse(string a)
```

- Function: reverse the string
- Return type: string type

Example

```
mysql> select reverse('palo');
+----------------+
| reverse('palo') |
+----------------+
| olap           |
+----------------+
```

Keywords

```
reverse
```

RPAD

Description

```
rpad(string str, int len, string pad)
```

- Function: return a string of length len (starting from the first letter) in str. If the len is greater than the length of str, then add pad characters behind str until the length of the string reaches len. If len is smaller than the length of str, this function is equivalent to truncating str string and will return only the string with length len.
- Return type: string type

Example

```
mysql> select rpad("hello", 10, 'xy');
+-------------------------+
| rpad('hello', 10, 'xy') |
+-------------------------+
| helloxyxyx              |
+-------------------------+
```

Keywords

```
rpad
```

RTRIM

Description

```
rtrim(string a)
```

- Function: remove the spaces that appear continuously in the right part of the parameter. Compare with ltrim () to find the difference in function. One is removed with the space before the string, and the other is removed with the space after the string.
- Return type: string type

Example

```
mysql> select rtrim('    today is friday    ');
+-------------------------------+
| rtrim('    today is friday    ') |
+-------------------------------+
|     today is friday             |
+-------------------------------+

mysql> select ltrim('    today is friday    ');
+-------------------------------+
| ltrim('    today is friday    ') |
+-------------------------------+
| today is friday                 |
+-------------------------------+
```

Keywords

```
rtrim
```

SPACE

Description

```
space(int n)
```

- Function: return a string of n spaces
- Return type: string type

Example

```
mysql> select space(10);
+------------+
| space(10)  |
+------------+
|            |
+------------+

mysql> select space(20);
+--------------------+
| space(20)          |
+--------------------+
|                    |
+--------------------+
```

Keywords

```
space
```

SPLIT_PART

Description

```
split_part(string str, string splitStr, int num)
```

- Function: divide the Str according to splitStr, and return the num value
- Return type: string type

Example

```
select split_part('12,31,45,232', ',', 3);
+----------------------------------+
| split_part('12,31,45,232', ',', 3) |
+----------------------------------+
| 45                               |
+----------------------------------+
```

Keywords

```
split_part
```

STARTS_WITH

Description

```
starts_with(string str, string strPrefix)
```

- Function: Judge whether str starts with strPrefix
- Return type: bool type

Example

```
mysql> select starts_with('baidu palo','palo');
+-----------------------------------+
| starts_with('baidu palo', 'palo') |
+-----------------------------------+
|                                 0 |
+-----------------------------------+

mysql> select starts_with('baidu palo','baidu');
+-----------------------------------+
| starts_with('baidu palo', 'baidu') |
+-----------------------------------+
|                                 1 |
+-----------------------------------+
```

Keywords

```
starts_with
```

STRLEFT,LEFT

Description

```
strleft(string a, int num_chars)

left(string a, int num_chars)
```

- Function: return the leftmost num_chars characters in a string.
- Return type: string type

Example

```
mysql> select strleft('palo@baidu',5);
+--------------------------+
| strleft('palo@baidu', 5) |
+--------------------------+
| palo@                    |
+--------------------------+

mysql> select left('palo@baidu',4);
+----------------------+
| left('palo@baidu', 4) |
+----------------------+
| palo                 |
+----------------------+
```

Keywords

```
strleft,left
```

STRRIGHT,RIGHT

Description

```
strright(string a, int num_chars)

right(string a, int num_chars)
```

- Function: return the rightmost num_chars characters in a string.
- Return type: string type

Example

```
mysql> select strright('palo@baidu',5);
+---------------------------+
| strright('palo@baidu', 5) |
+---------------------------+
| baidu                     |
+---------------------------+

mysql> select right('palo@baidu',6);
+-----------------------+
| right('palo@baidu', 6) |
+-----------------------+
| @baidu                |
+-----------------------+
```

Keywords

```
strright,right
```

SUBSTR,SUBSTRING

Description

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

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

- Function: it is a function for getting substring, which returns the partial string with length len from start in the string described by the first parameter. The first letter is subscripted with 1.
- Return type: string type

Example

```
mysql> select substring('baidupalo',6);
+--------------------------+
| substring('baidupalo', 6) |
+--------------------------+
| palo                     |
+--------------------------+
```

Keywords

```
substr,substring
```

TRIM

Description

```
trim(string a)
```

- Function: remove the continuous spaces in the right part and the continuous spaces in the left part of the parameter. The function has the same effect with that of using ltrim () and rtrim () at the same time.
- Return type: string type

Example

```
mysql> select trim('   today is friday   ');
+------------------------------+
| trim('   today is friday   ') |
+------------------------------+
| today is friday              |
+------------------------------+
```

Keywords

```
trim
```

UPPER,UCASE

Description

```
upper(string a)

ucase(string a)
```

- Function: convert all letters in a string to uppercase.
- Return type: string type

Example

```
mysql> select upper('toDAY Is FridAy');
+-------------------------+
| upper('toDAY Is FridAy') |
+-------------------------+
| TODAY IS FRIDAY         |
+-------------------------+

mysql> select ucase('palo');
+--------------+
| ucase('palo') |
+--------------+
| PALO         |
+--------------+
```

Keywords

```
upper, ucase
```

Conditional Function

Palo supports the following conditional functions:

1.case
2.coalesce
3.if
4.ifnull
5.nullif

CASE

Description

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

- Function: compare the expression with several possible values and return the corresponding results when matching
- Return type: 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                                                   |
+-----------------------------------------------------------------+

mysql> select case when tiny_column = 1 then "tiny_column=1" when tiny_column = 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                                                   |
+-----------------------------------------------------------------+
```

Keywords

```
case
```

COALESCE

Description

```
coalesce( expression,value1,value2…… ,valuen)
```

- Function: return the first non-empty expression in all parameters including expression. The expression is an expression to be detected, and the number of parameters after it is variable.
- Return type: type of result returned after matching

Example

```
mysql> select coalesce(NULL, '1111', '0000');
+------------------------------+
| coalesce(NULL, '1111', '0000') |
+------------------------------+
| 1111                         |
+------------------------------+
```

Keywords

```
coalesce
```

IF

Description

```
if(boolean condition, type ifTrue, type ifFalseOrNull)
```

- Function: test an expression and return the corresponding result according to 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                                |
+-------------------------------------+
```

Keywords

```
if
```

IFNULL

Description

```
ifnull(expr1, expr2)
```

- Function: test an expression and return the second parameter if the expression is NULL, otherwise return the first parameter.
- Return type: type of the first parameter

Example

```
mysql> select ifnull(1,0);
+------------+
| ifnull(1, 0) |
+------------+
|          1 |
+------------+

mysql> select ifnull(null,10);
+----------------+
| ifnull(NULL, 10) |
+----------------+
|             10 |
+----------------+
```

Keywords

```
ifnull
```

NULLIF

Description

```
nullif(expr1, expr2)
```

- Function: If the two parameters are equal, then it will return NULL. Otherwise, return the value of the first parameter. It has the same effect as the following CASE WHEN.
- Return type: expr1 type or NULL

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

Example

```
mysql> select nullif(1,1);
+-------------+
| nullif(1, 1) |
+-------------+
|        NULL |
+-------------+

mysql> select nullif(1,0);
+-------------+
| nullif(1, 0) |
+-------------+
|           1 |
+-------------+
```

Keywords

```
nullif
```

HLL Function

This article mainly introduces the built-in functions related to HLL(HyperLogLog) type.

- Scalar function
  1. hll_cardinality
  2. hll_hash
  3. hll_empty
- Aggregate function
  1. hll_union,hll_raw_agg
  2. hll_union_agg

HLL_CARDINALITY

Description

```
bigint hll_cardinality(hll a)
```

- Function: calculate the cardinality of a single HLL type value.
- Returned value: bigint type.

Example

```
mysql> select hll_cardinality(v1) from tbl;
+----------------------+
| hll_cardinality(`v1`) |
+----------------------+
|                    3 |
+----------------------+
```

Keywords

```
hll_cardinality
```

HLL_HASH

Description

```
hll hll_hash(type a)
```

- Function: convert a numeric value to hll type. It is generally used in import, mapping values in source data to HLL column types in Palo table.

- Returned value: hll type.

Example

```
mysql> select hll_cardinality(hll_hash("a"));
+----------------------------+
| hll_cardinality(hll_hash('a')) |
+----------------------------+
|                          1 |
+----------------------------+
```

> As HLL type is binary type, it cannot be displayed on MySQL client end. Therefore, `hll_cardinality` is used to return the carbinality of HLL type for display.

Keywords

```
hll_hash
```

HLL_EMPTY

Description

```
hll hll_empty()
```

- Function: return an empty value of HLL type. Generally it is used in import, inserted with an empty HLL value.
- Returned value: hll type.

Example

```
mysql> select hll_cardinality(hll_empty());
+------------------------------+
| hll_cardinality(hll_empty()) |
+------------------------------+
|                            0 |
+------------------------------+
```

Keywords

```
hll_empty
```

HLL_UNION,HLL_RAW_AGG

Description

```
hll hll_union(hll a)
hll hll_raw_agg(hll a)
```

- Function: an aggregate function, which returns the union set of a group of HLL values. `hll_raw_agg` is the alias of `hll_union`.
- Returned value: hll type.

Example

```
mysql> select k1, hll_cardinality(hll_union(v1)) from tbl group by k1;
+------+---------------------------------+
| k1   | hll_cardinality(hll_union(`v1`)) |
+------+---------------------------------+
|    2 |                               4 |
|    1 |                               3 |
+------+---------------------------------+
```

Keywords

```
hll_union, hll_raw_agg
```

HLL_UNION_AGG

Description

```
bigint hll_union_agg(hll a)
```

- Function: an aggregate function, which returns the cardinality of the union set of HLL values, with effect equivalent to `hll_cardinality(hll_union(v1))`. It is recommended to use `hll_union_agg` directly, which is more efficient.
- Returned value: bigint type.

Example

```
mysql> select k1, hll_union_agg(v1) from tbl group by k1;
+------+---------------------+
| k1   | hll_union_agg(`v1`) |
+------+---------------------+
|    2 |                   4 |
|    1 |                   3 |
+------+---------------------+
```

Keywords

```
hll_union_agg
```

Generic Function

Palo supports the following generic functions:

1.sleep
2.version

SLEEP

Description

```
sleep(int n)
```

- Function: session sleeps for n seconds, and returns success or failure
- Returned value: bool type

Example

```
mysql> select sleep(3);
+----------+
| sleep(3) |
+----------+
|        1 |
+----------+
1 row in set (3.07 sec)
```

Keywords

```
sleep
```

VERSION

Description

```
version()
```

- Function: return the version number of the current MySQL. With this function, users are compatible with some MySQL clients, but it does not refer to the version of task components in Palo.
- Returned value: string type

Example

```
mysql> select version();
+-----------+
| version() |
+-----------+
| 5.1.0     |
+-----------+
1 row in set (0.02 sec)
```

Keywords

```
version
```

BITMAP Function

This article mainly introduces the built-in functions related to BITMAP type.

- Scalar function
  1. to_bitmap
  2. bitmap_hash
  3. bitmap_count
  4. bitmap_empty
  5. bitmap_or
  6. bitmap_and
  7. bitmap_xor
  8. bitmap_to_string
  9. bitmap_from_string
  10. bitmap_contains
  11. bitmap_has_any

- Aggregate function

1. bitmap_union
2. bitmap_union_int
3. bitmap_union_count
4. bitmap_intersect
5. intersect_count

TO_BITMAP

Description

```
BITMAP to_bitmap(int i)
```

- Function: convert an integer to a BITMAP type. It supports BIGINT type at most, and is mostly used for import, mapping the integer column in the source data to the BITMAP column in the Palo table.
- Returned value: BITMAP type

Example

```
mysql> select bitmap_to_string(to_bitmap("123"));
+----------------------------------+
| bitmap_to_string(to_bitmap('123')) |
+----------------------------------+
| 123                              |
+----------------------------------+

mysql> select bitmap_to_string(to_bitmap(1));
+-----------------------------+
| bitmap_to_string(to_bitmap(1)) |
+-----------------------------+
| 1                           |
+-----------------------------+
```

> As BITMAP is a binary type, it cannot be displayed on MySQL client. Therefore, `bitmap_to_string` is used for the visual display of results.

Keywords

```
to_bitmap, bitmap
```

BITMAP_HASH

Description

```
BITMAP bitmap_hash(string s)
```

- Function: Map a string to 32-bit integer by Hash algorithm, and then convert it to BITMAP type. It is mostly used for import, mapping the non-integer column in the source data to the BITMAP column in the Palo table. As it adopts Hash algorithm, it might have Hash conflicts. That means, different strings may produce the same BITMAP value. So it can only be used for approximate calculation.
- Returned value: BITMAP type

Example

```
mysql> select bitmap_to_string(bitmap_hash("abc"));
+-----------------------------------+
| bitmap_to_string(bitmap_hash('abc')) |
+-----------------------------------+
| 3409700625                        |
+-----------------------------------+
```

Keywords

```
bitmap_hash, bitmap
```

BITMAP_COUNT

Description

```
bigint bitmap_count(bitmap s)
```

-
- Count the number of 1 in a BITMAP.
- Returned value: bigint type.

Example

```
mysql> select bitmap_count(bitmap_from_string("1,2,3"));
+-------------------------------------------+
| bitmap_count(bitmap_from_string('1,2,3')) |
+-------------------------------------------+
|                                         3 |
+-------------------------------------------+

mysql> select bitmap_count(bitmap_from_string("1,2,3,4"));
+-------------------------------------------+
| bitmap_count(bitmap_from_string('1,2,3,4')) |
+-------------------------------------------+
|                                         4 |
+-------------------------------------------+
```

Keywords

```
bitmap_count, bitmap
```

BITMAP_EMPTY

Description

```
bitmap bitmap_empty()
```

- Function: return an empty BITMAP. It is generally used for import, generating an empty bitmap.
- Returned value: bitmap type.

Example

```
mysql> select bitmap_to_string(bitmap_empty());
+--------------------------------+
| bitmap_to_string(bitmap_empty()) |
+--------------------------------+
|                                |
+--------------------------------+
```

Keywords

```
bitmap_empty, bitmap
```

BITMAP_OR

Description

```
bitmap bitmap_or(bitmap a, bitmap b)
```

- Function: return the union set of two bitmaps.
- Returned value: bitmap type.

Example

```
mysql> select bitmap_to_string(bitmap_or(bitmap_from_string("1,2,3,4"), bitmap_from_string("4,5,6")));
+-----------------------------------------------------------------------+
| bitmap_to_string(bitmap_or(bitmap_from_string('1,2,3,4'), bitmap_from_string('4,5,6'))) |
+-----------------------------------------------------------------------+
| 1,2,3,4,5,6                                                           |
+-----------------------------------------------------------------------+
```

Keywords

```
bitmap_or, bitmap
```

BITMAP_AND

Description

```
bitmap bitmap_and(bitmap a, bitmap b)
```

- Function: return the intersection set of two bitmaps.
- Returned value: bitmap type.

Example

```
mysql> select bitmap_to_string(bitmap_and(bitmap_from_string("1,2,3,4"), bitmap_from_string("4,5,6")));
+-----------------------------------------------------------------------+
| bitmap_to_string(bitmap_and(bitmap_from_string('1,2,3,4'), bitmap_from_string('4,5,6'))) |
+-----------------------------------------------------------------------+
| 4                                                                     |
+-----------------------------------------------------------------------+
```

Keywords

bitmap_and, bitmap

BITMAP_XOR

Description

```
bitmap bitmap_xor(bitmap a, bitmap b)
```

- Function: return the Exclusive OR result of two bitmaps.
- Returned value: bitmap type.

Example

```
mysql> select bitmap_to_string(bitmap_xor(bitmap_from_string("1,2,3,4"), bitmap_from_string("4,5,6")));
+-----------------------------------------------------------------------------+
| bitmap_to_string(bitmap_and(bitmap_from_string('1,2,3,4'), bitmap_from_string('4,5,6'))) |
+-----------------------------------------------------------------------------+
| 1,2,3,5,6                                                                   |
+-----------------------------------------------------------------------------+
```

Keywords

bitmap_xor, bitmap

BITMAP_TO_STRING

Description

```
string bitmap_to_string(bitmap a)
```

- Function: return the Exclusive OR result of two bitmaps. For example, when the first bit and the third bid of a bitmap are 1, then return `1,3`.
- Returned value: string.

Example

```
mysql> select bitmap_to_string(bitmap_from_string("4,5,6"));
+-------------------------------------------+
| bitmap_to_string(bitmap_from_string('4,5,6')) |
+-------------------------------------------+
| 4,5,6                                     |
+-------------------------------------------+
```

Keywords

```
bitmap_to_string, bitmap
```

BITMAP_FROM_STRING

Description

```
bitmap bitmap_from_string(string a)
```

- Function: parse a string and return a bitmap. The string is a list of values separated by comma, for example `1,200,301`
- Returned value: bitmap type.

Example

```
mysql> select bitmap_to_string(bitmap_from_string("4, 5, 6"));
+-------------------------------------------+
| bitmap_to_string(bitmap_from_string('4,5,6')) |
+-------------------------------------------+
| 4,5,6                                     |
+-------------------------------------------+
```

Keywords

```
bitmap_from_string, bitmap
```

BITMAP_CONTAINS

Description

```
boolean bitmap_contains(bitmap a, bigint b)
```

- Function: judge whether a bitmap contains the value specified.
- Returned value: bool type.

Example

```
mysql> select bitmap_contains(bitmap_from_string("4, 5 ,6"), 4);
+-----------------------------------------------+
| bitmap_contains(bitmap_from_string('4, 5 ,6'), 4) |
+-----------------------------------------------+
|                                             1 |
+-----------------------------------------------+

mysql> select bitmap_contains(bitmap_from_string("4, 5 ,6"), 7);
+-----------------------------------------------+
| bitmap_contains(bitmap_from_string('4, 5 ,6'), 7) |
+-----------------------------------------------+
|                                             0 |
+-----------------------------------------------+
```

Keywords

```
bitmap_contains, bitmap
```

BITMAP_HAS_ANY

Description

```
boolean bitmap_has_any(bitmap a, bitmap b)
```

- Function: Judge whether two bitmaps intersect.
- Returned value: bool type.

Example

```
mysql> select bitmap_has_any(bitmap_from_string("1,2,3,4"), bitmap_from_string("4,5,6"));
+-----------------------------------------------------------------+
| bitmap_has_any(bitmap_from_string('1,2,3,4'), bitmap_from_string('4,5,6')) |
+-----------------------------------------------------------------+
|                                                               1 |
+-----------------------------------------------------------------+

mysql> select bitmap_has_any(bitmap_from_string("1,2,3"), bitmap_from_string("4,5,6"));
+-----------------------------------------------------------------+
| bitmap_has_any(bitmap_from_string('1,2,3'), bitmap_from_string('4,5,6')) |
+-----------------------------------------------------------------+
|                                                               0 |
+-----------------------------------------------------------------+
```

Keywords

```
bitmap_has_any, bitmap
```

BITMAP_UNION

Description

```
bitmap bitmap_union(bitmap a)
```

- Function: an aggregate function, which returns the union set of a set of bitmap.
- Returned value: bitmap type.

Example

```
mysql> select k1, bitmap_to_string(bitmap_union(v1)) from tbl1 group by k1;
+------+--------------------------------+
| k1   | bitmap_to_string(bitmap_union(`v1`)) |
+------+--------------------------------+
|    2 | 2,3,4                          |
|    1 | 1,2,3                          |
+------+--------------------------------+
```

Keywords

```
bitmap_union, bitmap
```

BITMAP_UNION_INT

Description

```
bigint bitmap_union_int(int a)
```

- Function: an aggregate function, which uses bitmap data structure to calculate the deduplication value of integer columns, and is equivalent to `count(distinct a)`. It supports the types of parameters TINYINT, SMALLINT, INT, BIGINT. The function can use the bitmap data structure and get the RemoveDuplicate value with less system resources.
- Returned value: bigint type.

Example

```
mysql> select bitmap_union_int(k1) from tbl1;
+------------------------+
| bitmap_union_int(`k1`) |
+------------------------+
|                      2 |
+------------------------+
```

Keywords

```
bitmap_union_int, bitmap
```

BITMAP_UNION_COUNT

Description

```
bigint bitmap_union_count(bitmap a)
```

- Function: an aggregate function, which returns the number of 1 in the union set of a set of bitmap, and is equivalent to `bitmap_count(bitmap_union(a))`. It is recommended to directly use bitmap_union_count` , which is more efficient.
- Returned value: bigint type.

Example

```
mysql> select k1, bitmap_union_count(v1) from tbl1 group by k1;
+------+-------------------------+
| k1   | bitmap_union_count(`v1`) |
+------+-------------------------+
|    2 |                       3 |
|    1 |                       3 |
+------+-------------------------+
```

Keywords

```
bitmap_union, bitmap
```

BITMAP_INTERSECT

Description

```
bitmap bitmap_intersect(bitmap a)
```

- Function: an aggregate function, which return the intersection of a set of bitmap.
- Returned value: bitmap type.

The function of this aggregate function corresponds to `bitmap_union`, with a slight difference with `bitmap_union` usage scenario.

At present, when building a table in Palo, the `bitmap_union` aggregation method must be specified for the bitmap type column (note that this refers to the **Aggregation Method**, not the **Aggregation Function**). And the aggregation mode is of practical value only when queried by an equivalent aggregation function. Therefore, for columns of bitmap type, quires as follows are meaningful:

```
select k1, bitmap_union(v1) from tbl group by k1;
```

And queries like the following are meaningless:

```
select k1, bitmap_intersect(v1) from tbl group by k1;
```

See the example for the specific usage of `bitmap_intersect` .

Example

The table structure is as follows:

```
 k1  INT
v1  BITMAP  BITMAP_UNION
```

When querying, firstly aggregate data by using subqueries through `bitmap_union` , then obtain the intersection at outer layer through `bitmap_intersect`:

```
mysql> select bitmap_to_string(bitmap_intersect(x)) from (select k1, bitmap_union(v1) x from tbl1 group by k1) a;
+----------------------------------------+
| bitmap_to_string(bitmap_intersect(`x`)) |
+----------------------------------------+
| 2,3                                    |
+----------------------------------------+
```

Keywords

```
bitmap_intersect, bitmap
```

INTERSECT_COUNT

Description

```
bigint intersect_count(bitmap a, column c, type cond1[, type cond2, ...])
```

- Function:

  This function is commonly used in business scenarios such as calculating retention. The first parameter is the bitmap column for which the retention calculation is to be performed. The second parameter is the column name for which intersection calculation is required. The variable-length parameter after that is a set of values of the column corresponding to the second parameter.

  The variable-length parameter after that is a set of values of the column corresponding to the second parameter.

  `bitmap_count` + `bitmap_intersect` + `bitmap_union` + `where`

- Returned value: bigint type.

Example

Calculate the user retention on 2020-10-01 and 2020-10-02. The table structure is as follows:

```
 dt      DATETIME
user_id BITMAP
```

```
mysql> select intersect_count(user_id, dt, '2020-10-01', '2020-10-02'), intersect_count(user_id, dt, '2020-10-01')
from tbl where dt in ('2020-10-01', '2020-10-02');
+-----------------------------------------------------+-----------------------------------------+
| intersect_count(`user_id`, `dt`, '2020-10-01', '2020-10-02') | intersect_count(`user_id`, `dt`, '2020-10-01') |
+-----------------------------------------------------+-----------------------------------------+
|                                                   3 |                                       7 |
+-----------------------------------------------------+-----------------------------------------+
```

The above results show that, the number of visitors on 2020-10-01 was 7, and the number of visitors among the above visitors who visited again on 2020-10-02 was 3.

In which, `intersect_count(user_id, dt, '2020-10-01', '2020-10-02')` is equivalent to the following statements:

```
select bitmap_count(bitmap_intersect(b))
from
(
    select dt, bitmap_union(user_id) b from tbl2
    where dt in ('2020-10-01', '2020-10-02')
    group by dt
) t2
```

Keywords

```
intersect_count, bitmap
```

Format Conversion Function

The conversion functions of Palo format are as follows:

1.aes_encrypt
2.aes_decrypt
3.to_base64
4.from_base64
5.md5,md5sum

AES_ENCRYPT

Description

```
aes_encrypt(string str, string key)
```

- Function: Use key to encrypt str and return the encrypted result. This function uses AES algorithm to encrypt a given string with a key length of 128 bits, and locks the encrypted string with the password given in the second parameter. If one of the given parameters is null, the function will return a null value.
- Return type: string type

Example

Due to the problems with the display of encrypted characters, here we use `HEX()` function to encode the encrypted character string in hexadecimal system, and then extract it with `UNHEX()` when decrypting.

```
mysql> select hex(aes_encrypt("Palo is Great", "baidu"));
+------------------------------------------+
| hex(aes_encrypt('Palo is Great', 'baidu')) |
+------------------------------------------+
| 676EC1EDBB586B736A23257E0ED78C17         |
+------------------------------------------+
```

Keywords

```
aes_encrypt
```

AES_DECRYPT

Description

```
aes_decrypt(string str, string key)
```

- Function: use key to decrypt str, and return the decrypted result
- Return type: string type

Example

Decrypt the encrypted string with aes_decrypt, first extract the hexadecimal code with UNHEX (), and then decrypt it with decryption function.

```
mysql> select aes_decrypt(unhex('676EC1EDBB586B736A23257E0ED78C17'),'baidu');
+-----------------------------------------------------------------+
| aes_decrypt(unhex('676EC1EDBB586B736A23257E0ED78C17'), 'baidu') |
+-----------------------------------------------------------------+
| Palo is Great                                                   |
+-----------------------------------------------------------------+
```

Keywords

```
aes_decrypt
```

TO_BASE64

Description

```
to_base64(string str)
```

- Function: convert str to base64 format and return the result of base64
- Return type: string type

Example

```
mysql> select to_base64('palo');
+------------------+
| to_base64('palo') |
+------------------+
| cGFsbw==         |
+------------------+
```

Keywords

```
to_base64
```

FROM_BASE64

Description

```
from_base64(string str)
```

- Function: decrypt str in base64 format and return the decrypted result.
- Return type: string type

Example

```
mysql> select from_base64('cGFsbw==');
+------------------------+
| from_base64('cGFsbw==') |
+------------------------+
| palo                   |
+------------------------+
```

Keywords

```
from_base64
```

MD5, MD5SUM

Description

```
md5(string str)

md5sum(string str)
```

- Function: convert str to md5 format and return the result of md5
- Return type: string type

Example

```
mysql> select md5('palo');
+----------------------------------+
| md5('palo')                      |
+----------------------------------+
| b50347c6fa8e55d5b562fe0f1511d324 |
+----------------------------------+

mysql> select md5sum('palo');
+----------------------------------+
| md5sum('palo')                   |
+----------------------------------+
| b50347c6fa8e55d5b562fe0f1511d324 |
+----------------------------------+
```

Keywords

```
md5, md5sum
```

Time and Date Function

Palo supports two types of time: DATE and DATETIME.

- The format of DATE type is: "2020-10-10"
- The format of DATETIME is: "2020-10-10 11:10:06"

Palo supports the following date and time functions:

1.add_months
2.adddate
3.convert_tz
4.curdate,current_date
5.current_timestamp
6.curtime,current_time
7.date_add
8.date_sub
9.date_format
10.datediff
11.day,dayofmonth
12.dayname
13.dayofweek
14.dayofyear
15.days_add
16.days_diff
17.days_sub
18.extract
19.from_days
20.from_unixtime 21.unix_timestamp
22.utc_timestamp
23.hour
24.hours_add
25.hours_diff
26.hours_sub
27.localtime,localtimestamp
28.microseconds_add
29.microseconds_sub
30.minute
31.minutes_add
32.minutes_diff
33.minutes_sub
34.month
35.monthname
36.months_add
37.months_diff
38.months_sub
39.now
40.second
41.seconds_add
42.seconds_diff
43.seconds_sub
44.subdate
45.str_to_date
46.timediff

47.to_date
48.to_days
49.weeks_add
50.weekofyear
51.weeks_diff
52.weeks_sub
53.quarter
54.year
55.year_floor
56.years_add
57.years_diff
58.years_sub

ADD_MONTHS

Description

```
add_months(timestamp date, int months)

add_months(timestamp date, bigint months)
```

- Function: return a new date consisting of specified date and months, same as months_add()
- Return type: timestamp type

Example

If this day of this month does not exist in the target month, the result will be the last day of that month;if months in the parameter is negative, then calculate the previous month.

```
 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)
```

Keywords

```
add_months
```

ADDDATE

Description

```
 adddate(timestamp startdate, int days)

adddate(timestamp startdate, bigint days)
```

- Function: add 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 |
+-----------------------------+
```

Keywords

```
adddate
```

CONVERT_TZ

Description

```
convert_tz(timestamp date, string from, string to)
```

- Function: convert the time zone of the specified time
- Return type: timestamp type

Example

```
mysql> select convert_tz('2020-12-20 12:00:00','+00:00','+10:00');
+-----------------------------------------------+
| convert_tz('2020-12-20 12:00:00', '+00:00', '+10:00') |
+-----------------------------------------------+
| 2020-12-20 22:00:00                           |
+-----------------------------------------------+
1 row in set (0.09 sec)
```

Keywords

```
convert_tz
```

CURDATE,CURRENT_DATE

Description

```
curdate()
current_date()
```

- Function: get the current date
- Return type: timestamp type

Example

```
mysql> select curdate();
+------------+
| curdate()  |
+------------+
| 2020-12-25 |
+------------+
1 row in set (0.03 sec)

mysql> select current_date();
+--------------+
| current_date() |
+--------------+
| 2020-12-25     |
+--------------+
1 row in set (0.02 sec)
```

Keywords

```
curdate,current_date
```

CURRENT_TIMESTAMP

Description

```
current_timestamp()
```

- Function: it has the same function as the now () function and gets the current time
- Return type: timestamp type

Example

```
mysql> select current_timestamp();
+---------------------+
| current_timestamp() |
+---------------------+
| 2020-12-25 14:13:10 |
+---------------------+
1 row in set (0.03 sec)
```

Keywords CURRENT_TIMESTAMP

CURTIME,CURRENT_TIME

Description

```
curtime()
current_time()
```

- Function: get the current time, excluding the date field
- Return type: timestamp type

Example

```
mysql> select curtime();
+-----------+
| curtime() |
+-----------+
| 14:24:07  |
+-----------+
1 row in set (0.02 sec)

mysql> select current_time();
+---------------+
| current_time() |
+---------------+
| 14:24:22      |
+---------------+
1 row in set (0.08 sec)
```

Keywords

```
curtime,current_time
```

DATE_ADD

Description

```
date_add(timestamp startdate, int days)
```

- Function: add the specified number of days to the TIMESTAMP value, the first parameter can be a string, and if the string conforms to the format of TIMESTAMP data type, the string will be automatically converted to TIMESTAMP type. The second parameter is the time interval.
- Return type: timestamp type

Example

```
mysql> select date_add('2020-12-25',20);
+-----------------------------------+
| date_add('2020-12-25 00:00:00', 20) |
+-----------------------------------+
| 2021-01-14 00:00:00               |
+-----------------------------------+
1 row in set (0.03 sec)
```

Keywords

```
date_add
```

DATE_SUB

Description

```
date_sub(timestamp startdate, int days)
```

- Function: minus the specified number of days to the TIMESTAMP value. The first parameter can be a string, and if the string conforms to the format of TIMESTAMP data type, the string will be automatically converted to TIMESTAMP type. The second parameter is the time interval.
- Return type: timestamp type

Example

```
mysql> select date_sub('2020-12-25',20);
+-----------------------------------+
| date_sub('2020-12-25 00:00:00', 20) |
+-----------------------------------+
| 2020-12-05 00:00:00                |
+-----------------------------------+
1 row in set (0.02 sec)
```

Keywords

```
date_sub
```

DATE_FORMAT

Description

```
date_format(timestamp day, string fmt)
```

- Function: convert the date type into a string by the format type, currently supporting the string of max 128 bytes. If the return length exceeds 128, then return NULL.
- Return type: string type
- The meaning of format is as follows:

```
 %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)
```

Keywords

```
date_format
```

DATEDIFF

Description

```
datediff(string enddate, string startdate)
```

- Function: return the difference in the number of days between two dates
- Return type: int type

Example

```
mysql> select datediff('2020-12-25','2019-11-20');
+-----------------------------------------------------+
| datediff('2020-12-25 00:00:00', '2019-11-20 00:00:00') |
+-----------------------------------------------------+
|                                                 401 |
+-----------------------------------------------------+
1 row in set (0.03 sec)
```

Keywords

```
datediff
```

DAY,DAYOFMONTH

Description

```
day(string date)
dayofmonth(string date)
```

- Function: return the day field in a date
- Return type: int type

Example

```
 mysql> select day('2020-12-25');
+---------------------------+
| day('2020-12-25 00:00:00') |
+---------------------------+
|                        25 |
+---------------------------+
1 row in set (0.02 sec)

mysql> select dayofmonth('2020-12-25');
+-----------------------------------+
| dayofmonth('2020-12-25 00:00:00') |
+-----------------------------------+
|                                25 |
+-----------------------------------+
1 row in set (0.07 sec)
```

Keywords day,dayofmonth

DAYNAME

Description

```
 dayname(timestamp date)
```

- Function: return the specified day of the week (English)
- Return type: string type

Example

```
 mysql>  select dayname('2020-12-25');
+-----------------------------+
| dayname('2020-12-25 00:00:00') |
+-----------------------------+
| Friday                      |
+-----------------------------+
1 row in set (0.04 sec)
```

Keywords

```
dayname
```

DAYOFWEEK

Description

```
dayofweek(timestamp date)
```

- Function: return the specified day of the week (number)

> Notes: 1=Sunday, 2=Monday, 3=Tuesday, 4=Wednesday, 5=Thursday, 6=Friday, 7=Saturday.

- Return type: int type

Example

```
mysql> select dayofweek('2020-12-25');
+-----------------------------------+
| dayofweek('2020-12-25 00:00:00') |
+-----------------------------------+
|                                 6 |
+-----------------------------------+
1 row in set (0.07 sec)
```

Keywords

```
dayofweek
```

DAYOFYEAR

Description

```
dayofyear(timestamp date)
```

- Function: return the specified day of the current year
- Return type: int type

Example

```
mysql> select dayofyear('2020-12-25');
+-----------------------------------+
| dayofyear('2020-12-25 00:00:00') |
+-----------------------------------+
|                               360 |
+-----------------------------------+
1 row in set (0.02 sec)
```

Keywords

```
dayofyear
```

DAYS_ADD

Description

```
days_add(timestamp startdate, int days)

days_add(timestamp startdate, bigint days)
```

- Function: add the specified number of days to startdate, which is similar to the date_add function, except that the parameter of this function is TIMESTAMP type instead of string type.

- Return type: timestamp type

Example

```
mysql> select days_add('2020-12-25',10);
+-----------------------------------+
| days_add('2020-12-25 00:00:00', 10) |
+-----------------------------------+
| 2021-01-04 00:00:00                 |
+-----------------------------------+
1 row in set (0.02 sec)
```

Keywords

```
days_add
```

DAYS_DIFF

Description

```
days_diff(timestamp enddate, timestamp startdate)
```

- Function: the difference between the start time and the end time
- Return type: int type

Example

```
mysql> select days_diff('2020-12-25','2020-10-1');
+-----------------------------------------------------+
| days_diff('2020-12-25 00:00:00', '2020-10-01 00:00:00') |
+-----------------------------------------------------+
|                                                  85 |
+-----------------------------------------------------+
1 row in set (0.06 sec)
```

Keywords

```
days_diff
```

DAYS_SUB

Description

```
days_sub(timestamp startdate, int days)

days_sub(timestamp startdate, bigint days)
```

- Function: subtract specified number of days from startdate; which is similar to the date_sub function, except that the parameter of this function is TIMESTAMP type instead of string type.
- Return type: timestamp type

Example

```
mysql> select days_sub('2020-12-25',20);
+----------------------------------+
| days_sub('2020-12-25 00:00:00', 20) |
+----------------------------------+
| 2020-12-05 00:00:00              |
+----------------------------------+
1 row in set (0.03 sec)
```

Keywords

```
days_sub
```

EXTRACT

Description

```
extract(unit FROM timestamp)
```

- Function: extract the value of timestamp in a specified unit. The unit can 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 |
+---------------------+-----------+------------+
| 2020-12-25 18:27:13 |      2020 |         12 |
+---------------------+-----------+------------+
1 row in set (0.02 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 |
+---------------------+----------+-----------+
| 2020-12-25 18:28:26 |       25 |        18 |
+---------------------+----------+-----------+
1 row in set (0.02 sec)
```

Keywords

```
extract
```

FROM_DAYS

Description

```
from_days(int days)
```

- Function: return the date from 0000-00-00 for the specified number of days backward
- Return type: timestamp

Example

```
mysql> select from_days(10000);
+-----------------+
| from_days(10000) |
+-----------------+
| 0027-05-19      |
+-----------------+
1 row in set (0.09 sec)
```

Keywords

```
from_days
```

FROM_UNIXTIME

Description

```
from_unixtime(bigint unixtime,[ string format])
```

- Function: convert unix time (seconds elapsed since January 1, 1970) to a date type in the corresponding format
- Return type: string type
- Instructions for use: the current date format is case-sensitive, users should especially distinguish between lowercase m (representing minutes) and uppercase M (representing months). The complete pattern of the date string is "yyyy-MM-dd HH: mm: Ss.SSSSSS ",or it can only contain some of the fields.

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 in combination to convert the timestamp type into a string in a specified format.

```
mysql> select from_unixtime(unix_timestamp(now()), 'yyyy-MM-dd');
+----------------------------------------------+
| from_unixtime(unix_timestamp(now()), '%Y-%m-%d') |
+----------------------------------------------+
| 2020-12-25                                   |
+----------------------------------------------+
1 row in set (0.02 sec)

mysql> select from_unixtime(unix_timestamp(now()), '%Y %D %M');
+----------------------------------------------+
| from_unixtime(unix_timestamp(now()), '%Y %D %M') |
+----------------------------------------------+
| 2020 25th December                           |
+----------------------------------------------+
1 row in set (0.03 sec)
```

UNIX_TIMESTAMP

Description

```
unix_timestamp()

unix_timestamp(string datetime)

unix_timestamp(string datetime, string format)

unix_timestamp(timestamp datetime)
```

- Function: return the timestamp of the current time (relative to the number of seconds of January 1, 1970) or convert it from a specified date and time to a timestamp. The timestamp returned is a timestamp relative to Greenwich Mean Time Zone.
- Return type: bigint type

Example

```
mysql> select unix_timestamp();
+------------------+
| unix_timestamp() |
+------------------+
|       1608896139 |
+------------------+
1 row in set (0.03 sec)
```

Keywords

```
unix_timestamp
```

UTC_TIMESTAMP

Description

```
utc_timestamp()
```

- Function: return the time in the current Greenwich Mean Time Zone
- Return type: timestamp

Example

```
mysql> select utc_timestamp();
+---------------------+
| utc_timestamp()     |
+---------------------+
| 2020-12-25 11:39:30 |
+---------------------+
1 row in set (0.02 sec)
```

Keywords

```
utc_timestamp
```

HOUR

Description

```
hour(string date)
```

- Function: return the hour field of the date expressed by a string
- Return type: int type

Example

```
mysql> select hour('2020-12-25 23:46');
+--------------------------+
| hour('2020-12-25 23:46') |
+--------------------------+
|                       23 |
+--------------------------+
1 row in set (0.02 sec)
```

Keywords hour

HOURS_ADD

Description

```
 hours_add(timestamp date, int hours)

hours_add(timestamp date, bigint hours)
```

- Function: return the time of specified date plus the number of hours
- Return type: timestamp

Example

```
 mysql> select hours_add('2020-12-25 18:00', 24);
+---------------------------------+
| hours_add('2020-12-25 18:00', 24) |
+---------------------------------+
| 2020-12-26 18:00:00             |
+---------------------------------+
1 row in set (0.06 sec)

mysql> select hours_add(now(), 24);
+---------------------+
| hours_add(now(), 24) |
+---------------------+
| 2020-12-26 19:45:40  |
+---------------------+
1 row in set (0.02 sec)
```

Keywords

```
 hours_add
```

HOURS_DIFF

Description

```
hours_diff(timestamp enddate, timestamp startdate)
```

- Function: the difference between the start time and the end time, in days
- Return type: int type

Example

```
mysql> select hours_diff('2020-12-26 08:00','2020-12-25 20:00');
+-----------------------------------------------+
| hours_diff('2020-12-26 08:00', '2020-12-25 20:00') |
+-----------------------------------------------+
|                                            12 |
+-----------------------------------------------+
1 row in set (0.04 sec)
```

Keywords

```
hours_diff
```

HOURS_SUB

Description

```
hours_sub(timestamp date, int hours)

hours_sub(timestamp date, bigint hours)
```

- Function: return the time of specified date subtracting plus the number of hours

- Return type: timestamp

Example

```
mysql> select hours_sub('2020-12-25 20:00',12);
+-----------------------------------+
| hours_sub('2020-12-25 20:00', 12) |
+-----------------------------------+
| 2020-12-25 08:00:00               |
+-----------------------------------+
1 row in set (0.05 sec)
```

Keywords hours_sub

LOCALTIME,LOCALTIMESTAMP

Description

```
localtime ()

localtimestamp()
```

- Function: it has the same function as the now () function and gets the current time
- Return type: timestamp type

Example

```
mysql> select localtime();
+---------------------+
| localtime()         |
+---------------------+
| 2020-12-25 19:52:56 |
+---------------------+
1 row in set (0.04 sec)

mysql> select localtimestamp();
+---------------------+
| localtimestamp()    |
+---------------------+
| 2020-12-25 19:53:10 |
+---------------------+
1 row in set (0.02 sec)
```

Keywords localtime,localtimestamp

MICROSECONDS_ADD

Description

```
microseconds_add(timestamp date, int microseconds)

microseconds_add(timestamp date, bigint microseconds)
```

- Function: return the time of specified date plus the number of microseconds
- Return type: timestamp

Example

```
mysql> select microseconds_add('2020-12-25',1000000);
+-----------------------------------------------+
| microseconds_add('2020-12-25 00:00:00', 1000000) |
+-----------------------------------------------+
| 2020-12-25 00:00:01                           |
+-----------------------------------------------+
1 row in set (0.02 sec)
```

Keywords

```
microseconds_add
```

MICROSECONDS_SUB

Description

```
microseconds_sub(timestamp date, int microseconds)

microseconds_sub(timestamp date, bigint microseconds)
```

- Function: return the time of specified date subtracting the number of microseconds
- Return type: timestamp

Example

```
mysql> select microseconds_sub('2020-12-25',1000000);
+----------------------------------------------+
| microseconds_sub('2020-12-25 00:00:00', 1000000) |
+----------------------------------------------+
| 2020-12-24 23:59:59                          |
+----------------------------------------------+
1 row in set (0.03 sec)
```

Keywords

```
microseconds_sub
```

MINUTE

Description

```
minute(string date)
```

- Function: return the minutes field of the date expressed by a string
- Return type: int type

Example

```
mysql> select minute('2020-12-25 20:25:35');
+-----------------------------+
| minute('2020-12-25 20:25:35') |
+-----------------------------+
|                          25 |
+-----------------------------+
1 row in set (0.04 sec)
```

Keywords minute

MINUTES_ADD

Description

```
minutes_add(timestamp date, int minutes)

minutes_add(timestamp date, bigint minutes)
```

- Function: return the time of specified date plus the number of minutes
- Return type: timestamp

Example

```
mysql> select minutes_add('2020-12-25 20:00:00',25);
+---------------------------------------+
| minutes_add('2020-12-25 20:00:00', 25) |
+---------------------------------------+
| 2020-12-25 20:25:00                   |
+---------------------------------------+
1 row in set (0.02 sec)
```

Keywords minutes_add

MINUTES_DIFF

Description

```
minutes_diff(timestamp enddate, timestamp startdate)
```

- Function: the difference between the start time and the end time, in minutes
- Return type: int type

Example

```
mysql> select minutes_diff('2020-12-25 22:00:00','2020-12-25 21:00:00');
+-----------------------------------------------------------+
| minutes_diff('2020-12-25 22:00:00', '2020-12-25 21:00:00') |
+-----------------------------------------------------------+
|                                                        60 |
+-----------------------------------------------------------+
1 row in set (0.02 sec)
```

Keywords minutes_diff

MINUTES_SUB

Description

```
minutes_sub(timestamp date, int minutes)

minutes_sub(timestamp date, bigint minutes)
```

- Function: return the time of specified date subtracting the number of minutes
- Return type: timestamp

Example

```
mysql> select minutes_sub('2020-12-25 20:00:00',25);
+---------------------------------------+
| minutes_sub('2020-12-25 20:00:00', 25) |
+---------------------------------------+
| 2020-12-25 19:35:00                   |
+---------------------------------------+
1 row in set (0.02 sec)
```

Keywords minutes_sub

MONTH

Description

```
month(string date)
```

- Function: return the month field (number) of the date expressed by the string
- Return type: int type

Example

```
mysql> select month('2020-12-25');
+-----------------------------+
| month('2020-12-25 00:00:00') |
+-----------------------------+
|                          12 |
+-----------------------------+
1 row in set (0.02 sec)
```

Keywords month

MONTHNAME

Description

```
monthname(string date)
```

- Function: return the month field (English) of the date expressed by the string
- Return type: string type

Example

```
mysql> select monthname('2020-12-25');
+----------------------------------+
| monthname('2020-12-25 00:00:00') |
+----------------------------------+
| December                         |
+----------------------------------+
1 row in set (0.02 sec)
```

Keywords

MONTHS_ADD

Description

```
months_add(timestamp date, int months)

months_add(timestamp date, bigint months)
```

- Function: return the time of specified date plus the number of months
- Return type: timestamp

Example

```
mysql> select months_add('2020-12-25',2);
+-----------------------------------+
| months_add('2020-12-25 00:00:00', 2) |
+-----------------------------------+
| 2021-02-25 00:00:00               |
+-----------------------------------+
1 row in set (0.03 sec)
```

Keywords months_add

MONTHS_DIFF

Description

```
months_diff(timestamp enddate, timestamp startdate)
```

- Function: the difference between the start time and the end time, in months
- Return type: int type

Example

```
mysql> select months_diff('2020-12-25','2020-05-25');
+-----------------------------------------------------+
| months_diff('2020-12-25 00:00:00', '2020-05-25 00:00:00') |
+-----------------------------------------------------+
|                                                   7 |
+-----------------------------------------------------+
1 row in set (0.02 sec)
```

Keywords months_diff

MONTHS_SUB

Description

```
months_sub(timestamp date, int months)

months_sub(timestamp date, bigint months)
```

- Function: return the time of specified date subtracting the number of months
- Return type: timestamp

Example

```
mysql> select months_sub('2020-12-25',2);
+-------------------------------------+
| months_sub('2020-12-25 00:00:00', 2) |
+-------------------------------------+
| 2020-10-25 00:00:00                 |
+-------------------------------------+
1 row in set (0.02 sec)
```

Keywords months_sub

NOW

Description

```
now()
```

- Function: return the current date and time (the time zone of the East Eighth District)
- Return type: timestamp

Example

```
mysql> select now();
+---------------------+
| now()               |
+---------------------+
| 2020-12-25 20:36:03 |
+---------------------+
1 row in set (0.20 sec)
```

Keywords now

SECOND

Description

```
second(string date)
```

- Function: return the seconds field of the date expressed by a string
- Return type: int type

Example

```
mysql> select now() as right_now,
    -> second(now());
+---------------------+---------------+
| right_now           | second(now()) |
+---------------------+---------------+
| 2020-12-25 20:44:25 |            25 |
+---------------------+---------------+
1 row in set (0.03 sec)
```

Keywords second

SECONDS_ADD

Description

```
seconds_add(timestamp date, int seconds)

seconds_add(timestamp date, bigint seconds)
```

- Function: return the time of specified date plus the number of seconds
- Return type: timestamp

Example

```
mysql> select now() as right_now,
    -> seconds_add(now(),20);
+---------------------+---------------------+
| right_now           | seconds_add(now(), 20) |
+---------------------+---------------------+
| 2020-12-25 20:45:07 | 2020-12-25 20:45:27    |
+---------------------+---------------------+
1 row in set (0.02 sec)
```

Keywords seconds_add

SECONDS_DIFF

Description

```
seconds_diff(timestamp enddate, timestamp startdate)
```

- Function: the difference between the start time and the end time, in seconds
- Return type: int type

Example

```
mysql> select seconds_diff('2020-12-25 08:00:00','2020-12-25 07:00:00');
+-----------------------------------------------------+
| seconds_diff('2020-12-25 08:00:00', '2020-12-25 07:00:00') |
+-----------------------------------------------------+
|                                                3600 |
+-----------------------------------------------------+
1 row in set (0.05 sec)
```

Keywords

```
seconds_diff
```

SECONDS_SUB

Description

```
seconds_sub(timestamp date, int seconds)

seconds_sub(timestamp date, bigint seconds)
```

- Function: subtract the number of days from the startdate.

- Return type: timestamp

Example

```
mysql> select now() as right_now,
    -> seconds_sub(now(),20);
+---------------------+---------------------+
| right_now           | seconds_sub(now(), 20) |
+---------------------+---------------------+
| 2020-12-25 20:46:10 | 2020-12-25 20:45:50    |
+---------------------+---------------------+
1 row in set (0.03 sec)
```

Keywords seconds_sub

SUBDATE

Description

```
subdate(timestamp startdate, int days)

subdate(timestamp startdate, bigint days)
```

- Function: similar to the date_sub () function, but the first parameter of this function is the exact TIMESTAMP, not a string that can be converted into TIMESTAMP type.
- Return type: timestamp

Example

```
mysql> select subdate('2020-12-25',10);
+-----------------------------------+
| subdate('2020-12-25 00:00:00', 10) |
+-----------------------------------+
| 2020-12-15 00:00:00                |
+-----------------------------------+
1 row in set (0.02 sec)
```

Keywords subdate

STR_TO_DATE

Description

```
str_to_date(string str, string format)
```

- Function: translate str into timestamp type in the way specified by format, and return NULL if the conversion result is incorrect. The format supported is consistent with date_format.
- Return type: timestamp

Example

```
mysql> select str_to_date('20201225 1130','%Y%m%d %h%i');
+-----------------------------------------+
| str_to_date('20201225 1130', '%Y%m%d %h%i') |
+-----------------------------------------+
| 2020-12-25 11:30:00                      |
+-----------------------------------------+
1 row in set (0.03 sec)
```

Keywords str_to_date

TIMEDIFF

Description

```
timediff(string enddate, string startdate)
```

- Function: return the timestamp difference between two times
- Return type: int type

Example

```
mysql> select timediff('20201225','20201224');
+----------------------------------------------------+
| timediff('2020-12-25 00:00:00', '2020-12-24 00:00:00') |
+----------------------------------------------------+
| 24:00:00                                           |
+----------------------------------------------------+
1 row in set (0.02 sec)
```

Keywords timediff

TO_DATE

Description

```
to_date(timestamp)
```

- Function: return the date domain 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           |
+---------------------+-----------------------------+
| 2020-12-25 21:08:02 | The date today is 2020-12-25. |
+---------------------+-----------------------------+
1 row in set (0.03 sec)
```

Keywords to_date

TO_DAYS

Description

```
to_days(timestamp date)
```

- Function: return the number of days from 0000-00-00 to the specified date
- Return type: int type **Example**

```
mysql> select to_days('0000-12-25');
+----------------------+
| to_days('0000-12-25') |
+----------------------+
|                  359 |
+----------------------+
1 row in set (0.03 sec)
```

Keywords to_days

WEEK_ADD

Description

```
weeks_add(timestamp date, int weeks)

weeks_add(timestamp date, bigint weeks)
```

- Function: return the time of specified date plus the number of weeks
- Return type: timestamp

Example

```
mysql> select weeks_add('2020-12-25',3);
+-------------------------------------+
| weeks_add('2020-12-25 00:00:00', 3) |
+-------------------------------------+
| 2021-01-15 00:00:00                 |
+-------------------------------------+
1 row in set (0.03 sec)
```

Keywords weeks_add

WEEKOFYEAR

Description

```
weekofyear(timestamp date)
```

- Function: get the week of the year
- Return type: int

Example

```
mysql> select weekofyear('2020-12-25');
+-----------------------------------+
| weekofyear('2020-12-25 00:00:00') |
+-----------------------------------+
|                                52 |
+-----------------------------------+
1 row in set (0.03 sec)
```

Keywords weekofyear

WEEK_DIFF

Description

```
weeks_diff(timestamp enddate, timestamp startdate)
```

- Function: the difference between the start time and the end time, in weeks
- Return type: int type

Example

```
mysql> select weeks_diff('2020-12-25','2020-10-25');
+-----------------------------------------------------+
| weeks_diff('2020-12-25 00:00:00', '2020-10-25 00:00:00') |
+-----------------------------------------------------+
|                                                   8 |
+-----------------------------------------------------+
1 row in set (0.02 sec)
```

Keywords weeks_diff

WEEK_SUD

Description

```
weeks_sub(timestamp date, int weeks)

weeks_sub(timestamp date, bigint weeks)
```

- Function: return the time of specified date subtracting the number of weeks
- Return type: timestamp

Example

```
mysql> select weeks_sub('2020-12-25',3);
+----------------------------------+
| weeks_sub('2020-12-25 00:00:00', 3) |
+----------------------------------+
| 2020-12-04 00:00:00              |
+----------------------------------+
1 row in set (0.28 sec)
```

Keywords weeks_sub

QUARTER

Description

```
quarter(timestamp date)
```

- Function: return the quarter to which the specified date belongs
- Return type: int

Example

```
mysql> select quarter('2020-12-25');
+-----------------------------+
| quarter('2020-12-25 00:00:00') |
+-----------------------------+
|                           4 |
+-----------------------------+
1 row in set (0.02 sec)
```

Keywords quarter

YEAR

Description

```
year(string date)
```

- Function: return the year field of the date expressed by a string
- Return type: int type

Example

```
mysql> select year('2020-12-25');
+----------------------------+
| year('2020-12-25 00:00:00') |
+----------------------------+
|                       2020 |
+----------------------------+
1 row in set (0.02 sec)
```

Keywords year

YEAR_FLOOR

Description

```
year_floor(timestamp date)
```

- Function: round down the specified time, keep the field level to year
- Return type: timestamp

Example

```
mysql> select year_floor('2020-12-25');
+---------------------------------+
| year_floor('2020-12-25 00:00:00') |
+---------------------------------+
| 2020-01-01 00:00:00             |
+---------------------------------+
1 row in set (0.04 sec)
```

Keywords year_floor

YEARS_ADD

Description

```
years_add(timestamp date, int years)

years_add(timestamp date, bigint years)
```

- Function: return the time of specified date plus the number of years
- Return type: timestamp

Example

```
mysql> select years_add('2020-12-25',10);
+-------------------------------------+
| years_add('2020-12-25 00:00:00', 10) |
+-------------------------------------+
| 2030-12-25 00:00:00                 |
+-------------------------------------+
1 row in set (0.03 sec)
```

Keywords years_add

YEAR_DIFF

Description

```
years_diff(timestamp enddate, timestamp startdate)
```

- Function: the difference between the start time and the end time, in years
- Return type: int type

Example

```
mysql> select years_diff('2020-12-25','2011-04-30');
+---------------------------------------------------+
| years_diff('2020-12-25 00:00:00', '2011-04-30 00:00:00') |
+---------------------------------------------------+
|                                                 9 |
+---------------------------------------------------+
1 row in set (0.02 sec)
```

Keywords years_diff

YEAR_SUB

Description

```
years_sub(timestamp date, int years)

years_sub(timestamp date, bigint years)
```

- Function: return the specified date subtracting the number of years
- Return type: timestamp

Example

```
mysql> select years_sub('2020-12-25',10);
+-------------------------------------+
| years_sub('2020-12-25 00:00:00', 10) |
+-------------------------------------+
| 2010-12-25 00:00:00                 |
+-------------------------------------+
1 row in set (0.03 sec)
```

Keywords years_sub

Type Conversion Function

Palo supports the following types of conversion functions

1. cast

CAST

Description

```
cast(expr as type)
```

- Function: generally the conversion functions are used in combination with other functions, showing the conversion of expression to specified parameter type. Palo has strict definition of data type for parameter types of functions. For example, Palo will not automatically convert bigtint type to int type, or execute other conversions that may lose precision or overflow. Users can use cast function to convert column values or literal constants into other types required by function parameters.
- Return type: type after conversion.

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.                                  |
+-----------------------------------------------------------------+
```

Keywords

```
cast
```

Aggregate Function

The behavior of an aggregate function is to aggregate the results of multiple rows into one row.

Palo supports the following aggregate functions:

1. avg
2. count
3. approx_count_distinct
4. max
5. min
6. sum
7. group_concat
8. variance_samp
9. variance_pop
10. percentile_approx
11. topn

> For aggregate functions related to BITMAP type and HLL type, please refer to BITMAP Function and HLL Function.

AVG

Description

```
avg(numeric val)
```

- Function: the aggregate function returns the average value of numbers in a set. The function has only 1 parameter, which can be a column of number type, and the value it returns is the function of numbers, or the calculation result is the expression of numbers. Rows containing NULL value will be ignored. If the table is empty or all the parameters of AVG are NULL, then the function returns NULL. When `GROUP BY` clause is specified in a query, then the value of each `GROUP BY` will be returned in a result.
- Return type: double type

Example

```
mysql> select ss_ticket_number, avg(ss_sales_price) from store_sales group by ss_ticket_number;
+----------------+---------------------+
| ss_ticket_number | avg(`ss_sales_price`) |
+----------------+---------------------+
|          28818 |           41.041875 |
+----------------+---------------------+
```

Keywords

```
avg
```

COUNT

Description

```
count([distinct] col_name...)
```

- Function: the aggregat function returns the number of rows that meet the requirements, or the number of rows not containing NULL.
  - COUNT(*) count the rows containing NULL values.
  - COUNT(column_name) only counts the rows not containing NULL values.
  - Users can use the COUNT function and the DISTINCT operator at the same time, count(distinct col_name...) will remove the duplicate first, and then calculate the number of occurrences of the column combination.
- Return type: int type

Example

```
mysql> select count(distinct tiny_column, short_column) from small_table;
+-----------------------------------------+
| count(DISTINCT `tiny_column`, `short_column`) |
+-----------------------------------------+
|                                       2 |
+-----------------------------------------+
```

Keywords

```
count
```

APPROX_COUNT_DISTINCT

Description

```
approx_count_distinct(type col)
```

- Function: the aggregate function returns the deduplicate value of specified column. Unlike the `count(distinct)` method, the function uses the HyperLogLog algorithm to return duplicate values, which has some errors but has higher efficiency than `count(distinct)`.
- Return type: int type

Example

```
mysql> select approx_count_distinct(ss_ticket_number) from store_sales;
+----------------------------------------+
| approx_count_distinct(`ss_ticket_number`) |
+----------------------------------------+
|                                  46580 |
+----------------------------------------+
```

Keywords

```
approx_count_distinct
```

MAX

Description

```
max(type col)
```

- Function: this aggregate function returns the max value of numbers in a set. It is opposite to the min function. The function has only 1 parameter, which can be a column of number type, and the value it returns is the function of numbers, or the calculation result is the expression of numbers. Rows containing NULL value will be ignored. If the table is empty or all the parameters of MAX are NULL, then the function returns NULL. When `GROUP BY` clause is specified in a query, then the value of each `GROUP BY` will be returned in a result.
- Return type: same type as input parameters.

Example

```
mysql> select ss_ticket_number, max(ss_sales_price) from store_sales group by ss_ticket_number;
+----------------+---------------------+
| ss_ticket_number | max(`ss_sales_price`) |
+----------------+---------------------+
|          28818 |              136.42 |
+----------------+---------------------+
```

Keywords

```
max
```

MIN

Description

```
min(type col)
```

- Function: this aggregate function returns the min value of numbers in a set. It is opposite to the max function. The function has only 1 parameter, which can be a column of number type, and the value it returns is the function of numbers, or the calculation result is the expression of numbers. Rows containing NULL value will be ignored. If the table is empty or all the parameters of MIN are NULL, then the function returns NULL. When `GROUP BY` clause is specified in a query, then the value of each `GROUP BY` will be returned in a result.
- Return type: same type as input parameters.

Example

```
mysql> select ss_ticket_number, min(ss_sales_price) from store_sales group by ss_ticket_number;
+----------------+----------------------+
| ss_ticket_number | min(`ss_sales_price`) |
+----------------+----------------------+
|          28818 |                 0.91 |
+----------------+----------------------+
```

Keywords

```
min
```

SUM

Description

```
sum(numeric col)
```

- Function: the aggregate function returns the sum of all the values in a set. The function has only 1 parameter, which can be a column of number type, and the value it returns is the function of numbers, or the calculation result is the expression of numbers. Rows containing NULL value will be ignored. If the table is empty or all the parameters of MIN are NULL, then the function returns NULL. When `GROUP BY` clause is specified in a query, then the value of each `GROUP BY` will be returned in a result.
- Return type: if the parameter is integer, then return BIGINT, and if the parameter is floating point, return double.

Example

```
mysql> select ss_ticket_number, sum(ss_sales_price) from store_sales group by ss_ticket_number;
+----------------+----------------------+
| ss_ticket_number | sum(`ss_sales_price`) |
+----------------+----------------------+
|          28818 |           1971980.01 |
+----------------+----------------------+
```

Keywords

```
sum
```

GROUP_CONCAT

Description

```
group_count(col[, separator])
```

- Function: the aggregate function will return a string, which is a new string by connecting all strings in a set. If the user specifies a separator, the separator is used to connect strings of two adjacent rows.
- Return type: string type
- Instructions for use: by default, this function returns a string that covers all result sets. When GROUP BY clause is specified in a query, then the value of each GROUP BY will be returned in a result.

Example

```
 mysql> select * from tbl;
+------+------+
| k1   | v1   |
+------+------+
|    1 | a    |
|    1 | b    |
|    1 | c    |
+------+------+

mysql> select k1, group_concat(v1) from tbl group by k1;
+------+-------------------+
| k1   | group_concat(`v1`) |
+------+-------------------+
|    1 | a, b, c           |
+------+-------------------+

mysql> select k1, group_concat(v1, '|') from tbl group by k1;
+------+------------------------+
| k1   | group_concat(`v1`, '|') |
+------+------------------------+
|    1 | a|b|c                 |
+------+------------------------+
```

Keywords

```
 group_concat
```

VARIANCE,VAR_SAMP,VARIANCE_SAMP

Description

```
 variance(numeric val)
var_samp(numeric val)
variance_samp(numeric val)
```

- Function: this kind of aggregate function returns the sampel **Variance** of a set of numbers. This is a mathematical attribute representing the distance between the value and the average value. It acts on numeric types.`variance` and `var_samp` are the aliases of `variance_samp`.
- Return type: double type

Example

```
mysql> select ss_ticket_number, variance(ss_sales_price) from store_sales group by ss_ticket_number;
+------------------+---------------------------+
| ss_ticket_number | variance(`ss_sales_price`) |
+------------------+---------------------------+
|            28818 |            1378.408511209 |
+------------------+---------------------------+
```

Keywords

```
variance, var_samp, variance_samp
```

VAR_POP,VARIANCE_POP

Description

```
var_pop(numeric val)
variance_pop(numeric val)
```

- Function: this kind of aggregate function returns the population **Variance** of a set of numbers.This is a mathematical attribute representing the distance between the value and the average value. It acts on numeric types.`var_pop` is the alias of `variance_pop`.
- Return type: double type

Example

```
mysql> select ss_ticket_number, variance_pop(ss_sales_price) from store_sales group by ss_ticket_number;
+----------------+-------------------------------+
| ss_ticket_number | variance_pop(`ss_sales_price`) |
+----------------+-------------------------------+
|          28818 |              1378.408511134 |
+----------------+-------------------------------+
```

Keywords

```
var_pop, variance_pop
```

PERCENTILE_APPROX

Description

```
percentile_approx(numeric val, double percentile, double compression)
```

- Function: this kind of aggregate function adopts T-Digest algorithm, and returns the approximate value of a set of specified percentile values. The `percentile` should be between 0 and 1. The accuracy of the results can be controlled by `compression`, and the value range is 2014 - 10000. The greater the value is, the higher the accuracy is, the greater the memory overhead and time consumption are. The value is 2048 by default.
- Return type: double type

Example

```
mysql> select percentile_approx(query_time, 0.95), percentile_approx(query_time, 0.99) from tbl limit 10;
+-----------------------------------+-----------------------------------+
| percentile_approx(`query_time`, 0.95) | percentile_approx(`query_time`, 0.99) |
+-----------------------------------+-----------------------------------+
|                 30.994913101196289 |                 116.05957794189453 |
+-----------------------------------+-----------------------------------+

mysql> select `table`, percentile_approx(cost_time,0.99) from log_statis group by `table`;
+-----------------+-----------------------+
| table    | percentile_approx(`cost_time`, 0.99) |
+---------+-----------------------------------+
| test    |                             54.22 |
+---------+-----------------------------------+

mysql> select `table`, percentile_approx(cost_time,0.99, 4096) from log_statis group by `table`;
+-----------------+-------------------------------+
| table    | percentile_approx(`cost_time`, 0.99, 4096.0) |
+---------+-------------------------------------------+
| test    |                                 54.21 |
+---------+-------------------------------------------+
```

Keywords

```
percentile_approx, percentile
```

TOPN

Description

```
topn(expr, int top_num[, int space_expand_rate])
```

- Function: the topn function uses the Space-Saving algorithm to calculate the top_num frequent items in expr, and the results are frequent items and their occurrence times, which are approximate values. The `space_expand_rate` parameter is optional, which is used to set the number of counter used in Space-Saving algorithm. The greater the value of `space_expand_rate` is, the more accurate the result is. The default value is 50.

  ```
  counter numbers = top_num * space_expand_rate
  ```

- Return type: string of JSON format.

Example

```
mysql> select topn(time, 10) from tbl;
+-----------------------------------------------------------------------------------------------------------+
| topn(`time`, 10)                                                                                          |
+-----------------------------------------------------------------------------------------------------------+
| {"0":7894391,"1":3887461,"2":3792601,"6":3344590,"5":2394986,"7":1421491,"3":1046929,"29":982826,"30":674 |
| }                                                                                                         |
+-----------------------------------------------------------------------------------------------------------+

mysql> select topn(time, 10, 100) from tbl;
+-----------------------------------------------------------------------------------------------------------+
| topn(`time`, 10, 100)                                                                                     |
+-----------------------------------------------------------------------------------------------------------+
| {"0":7894592,"1":3887551,"2":3792700,"6":3344590,"5":2394986,"7":1421492,"3":1046977,"29":982826,"30":674 |
| }                                                                                                         |
+-----------------------------------------------------------------------------------------------------------+
```

Keywords

```
topn
```

Bit Operation Function

Palo supports the following bit operation functions:

1.bitand
2.bitnot
3.bitor
4.bitxor

BITAND

Desciption

```
bitand(integer_type a, same_type b)
```

- Function: Bitwise AND operation
- Return type: same as the input type

Example

```
mysql> select bitand(255, 32767); /* 0000000011111111 & 0111111111111111 */
+--------------------+
| bitand(255, 32767) |
+--------------------+
|                255 |
+--------------------+

mysql> select bitand(32767, 1); /* 0111111111111111 & 0000000000000001 */
+-------------------+
| bitand(32767, 1) |
+-------------------+
|                 1 |
+-------------------+

mysql> select bitand(32, 16); /* 00010000 & 00001000 */
+-----------------+
| bitand(32, 16) |
+-----------------+
|               0 |
+-----------------+

mysql> select bitand(12,5); /* 00001100 & 00000101 */
+----------------+
| bitand(12, 5) |
+----------------+
|              4 |
+----------------+

mysql> select bitand(-1,15); /* 11111111 & 00001111 */
+-----------------+
| bitand(-1, 15) |
+-----------------+
|              15 |
+-----------------+
```

Keywords

```
bitand
```

BITNOT

Desciption

```
bitnot(integer_type a)
```

- Function: BitNot operation
- Return type: same as the input type

Example

```
mysql> select bitnot(127); /* 01111111 -> 10000000 */
+------------+
| bitnot(127) |
+------------+
|       -128 |
+------------+

mysql> select bitnot(16); /* 00010000 -> 11101111 */
+-----------+
| bitnot(16) |
+-----------+
|       -17 |
+-----------+

mysql> select bitnot(0); /* 00000000 -> 11111111 */
+----------+
| bitnot(0) |
+----------+
|       -1 |
+----------+

mysql> select bitnot(-128); /* 10000000 -> 01111111 */
+-------------+
| bitnot(-128) |
+-------------+
|         127 |
+-------------+
```

Keywords

```
bitnot
```

BITOR

Desciption

```
bitor(integer_type a, same_type b)
```

- Function: BitOr operation
- Return type: same as the input type

Example

```
mysql> select bitor(1,4); /* 00000001 | 00000100 */
+------------+
| bitor(1, 4) |
+------------+
|          5 |
+------------+

mysql> select bitor(16,48); /* 00001000 | 00011000 */
+--------------+
| bitor(16, 48) |
+--------------+
|           48 |
+--------------+

mysql> select bitor(0,7); /* 00000000 | 00000111 */
+------------+
| bitor(0, 7) |
+------------+
|          7 |
+------------+
```

Keywords

```
bitor
```

BITXOR

Desciption

```
bitxor(integer_type a, same_type b)
```

- Function: BitOr operation
- Return type: same as the input type

Example

```
mysql> select bitxor(0,15); /* 00000000 ^ 00001111 */
+--------------+
| bitxor(0, 15) |
+--------------+
|           15 |
+--------------+

mysql> select bitxor(7,7); /* 00000111 ^ 00000111 */
+-------------+
| bitxor(7, 7) |
+-------------+
|           0 |
+-------------+

mysql> select bitxor(8,4); /* 00001000 ^ 00000100 */
+-------------+
| bitxor(8, 4) |
+-------------+
|          12 |
+-------------+

mysql> select bitxor(3,7); /* 00000011 ^ 00000111 */
+-------------+
| bitxor(3, 7) |
+-------------+
|           4 |
+-------------+
```

Keywords

```
bitxor
```

JSON Parsing Function

Currently, Palo supports three JSON parsing functions:

1.get_json_int
2.get_json_string
3.get_json_double

GET_JSON_INT

Description

```
get_json_int(VARCHAR json_str, VARCHAR json_path)
```

- Function: parse and get the integer content of the specified path in JSON string. The first parameter is json string, and the second parameter is the path in json.`json_path` must start with `$` symbol and use `.` as path separator. If the path contains `.`, it can be enclosed with double quotation marks. Use `[]` to indicate the array subscript, started with 0. The content of `path` cannot contain `"`, `[`, `]`. If the format of `json_string` is not correct, or the format of `json_path` is not correct, or no match item can be found, then return NULL.
- Return type: int type or NULL.

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 |
+-----------------------------------------------------------+
```

Keywords

```
GET_JSON_INT, JSON
```

GET_JSON_STRING

Description

```
get_json_string(VARCHAR json_str, VARCHAR json_path)
```

- Function: parse and obtain the string content of the specified path in JSON string. The`json_path` must start with `$` symbol and use `.` as path separator. If the path contains `.`, it can be enclosed with double quotation marks. Use `[]` to indicate the array subscript, started with 0. The content of `path` cannot contain `",[, ]`. If the format of `json_string` is not correct, or the format of`json_path` is not correct, or no match item can be found, then return NULL.
- Return type: string type or NULL.

Example

```
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                                                          |
+-----------------------------------------------------------------+
```

Keywords

```
GET_JSON_STRING, JSON
```

GET_JSON_DOUBLE

Description

```
get_json_double(VARCHAR json_str, VARCHAR json_path)
```

- Function: parse and obtain the floating point content of the specified path in JSON string. The `json_path` must start with `$`

as path separator. If the path contains . as path separator. If the path contains ., it can be enclosed with double quotation marks. Use [] to indicate the array subscript, started with 0. The content ofpath cannot contain ", [,]. If the format of json_string is not correct, or the format ofjson_path is not correct, or no match item can be found, then return NULL.

- Return type: double type or NULL.

Example

```
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 |
+-----------------------------------------------------------------+

mysql> select get_json_double('{"col1":100, "col2":"string", "col3":1.5}', "$.col5");
+-----------------------------------------------------------------+
| get_json_double('{"col1":100, "col2":"string", "col3":1.5}', '$.col5') |
+-----------------------------------------------------------------+
|                                                            NULL |
+-----------------------------------------------------------------+
```

Keywords

```
GET_JSON_DOUBLE, JSON
```

Mathematical Function

Palo supports the following mathematical functions:

1.sin
2.asin
3.tan
4.atan
5.cos
6.acos
7.abs
8.bin
9.ceil
10.floor
11.conv
12.degrees
13.e
14.exp
15.mod
16.fmod

17. pmod
18. greatest
19. least
20. hex
21. unhex
22. ln
23. dlog1
24. log
25. negative
26. positive
27. pi
28. pow
29. radians
30. rand
31. round
32. sign
33. sqrt
34. truncate

SIN

Description

```
sin(double a)
```

- Function: return the sine value of a
- Return type: double type

Example

```
mysql> select sin(1), sin(0.5 * pi());
+------------------+----------------+
| sin(1.0)         | sin(0.5 * pi()) |
+------------------+----------------+
| 0.8414709848078965 |              1 |
+------------------+----------------+
```

Keywords

```
sin
```

ASIN

Description

```
asin(double a)
```

- Function: arc-sin function, a must be between -1 and 1.
- Return type: double type

Example

```
mysql> select asin(0.8414709848078965), asin(2);
+-------------------------+----------+
| asin(0.8414709848078965) | asin(2.0) |
+-------------------------+----------+
|                       1 |      nan |
+-------------------------+----------+
```

Keywords

```
asin
```

TAN

Description

```
tan(double a)
```

- Function: arctangent value function
- Return type: double type

Example

```
mysql> select tan(pi()/4);
+--------------------+
| tan(pi() / 4.0)    |
+--------------------+
| 0.9999999999999989 |
+--------------------+
```

Keywords

```
tan
```

ATAN

Description

```
atan(double a)
```

- Function: arctangent value function
- Return type: double type

Example

```
mysql> SELECT ATAN(1.5574077246549023),ATAN(0);
+-------------------------+--------+
| ATAN(1.5574077246549023) | ATAN(0) |
+-------------------------+--------+
|                       1 |      0 |
+-------------------------+--------+
```

Keywords

```
atan
```

COS

Description

```
cos(double a)
```

- Function: return the cosine value of parameters
- Return type: double type

Example

```
mysql> select cos(1), cos(0), cos(pi());
+-------------------+---------+----------+
| cos(1.0)          | cos(0.0) | cos(pi()) |
+-------------------+---------+----------+
| 0.54030230586813977 |       1 |       -1 |
+-------------------+---------+----------+
```

Keywords

```
cos
```

ACOS

Description

```
acos(double a)
```

- Function: arc-cosine function, a must be between -1 and 1.
- Return type: double type

Example

```
mysql> select acos(2), acos(1), acos(-1);
+----------+----------+-------------------+
| acos(2.0) | acos(1.0) | acos(-1.0)        |
+----------+----------+-------------------+
|       nan |         0 | 3.1415926535897931 |
+----------+----------+-------------------+
```

Keywords

```
acos
```

ABS

Description

```
abs(numeric a)
```

- Function: return the absolute value of the parameter
- Return type: numeric type

Example

```
mysql> select abs(-1.2);
+----------+
| abs(-1.2) |
+----------+
|      1.2 |
+----------+

mysql> select abs(-10);
+---------+
| abs(-10) |
+---------+
|      10 |
+---------+
```

Keywords

```
abs
```

BIN

Description

```
bin(bigint a)
```

- Function: return the binary representation of an integer (that is, a sequence of zeros and ones)
- Return type: string type

Example

```
mysql> select bin(10);
+---------+
| bin(10) |
+---------+
| 1010    |
+---------+
```

Keywords

```
bin
```

CEIL,CEILING,DCEIL

Description

```
ceil(double a)
ceiling(double a)
dceil(double a)
```

- Function: return the smallest integer greater than or equal to the parameter
- Return type: int type

Example

```
mysql> select dceil(1.2), ceiling(1.2), ceil(1.2);
+-----------+-------------+----------+
| dceil(1.2) | ceiling(1.2) | ceil(1.2) |
+-----------+-------------+----------+
|          2 |           2 |        2 |
+-----------+-------------+----------+
```

Keywords

```
dceil, ceiling, ceil
```

FLOOR

Description

```
floor(double a)
dfloor(double a)
```

- Function: return the max integer less than or equal to this parameter
- Return type: int type

Example

```
mysql> select floor(2.9);
+-----------+
| floor(2.9) |
+-----------+
|          2 |
+-----------+

mysql> select dfloor(2.9);
+------------+
| dfloor(2.9) |
+------------+
|           2 |
+------------+
```

Keywords

```
floor, dfloor
```

CONV

Description

```
conv(bigint num, int from_base, int to_base)
conv(string num,int from_base, int to_base)
```

- Function: it is a function for converting number systems, which returns the string form of an integer in a specific number system. The input parameters can be integer strings. To convert the return value of a function into an integer, `CAST` function can be used.
- Return type: string type

Example

```
mysql> select conv(64,10,8);
+----------------+
| conv(64, 10, 8) |
+----------------+
| 100            |
+----------------+

mysql> select cast(conv('fe', 16, 10) as int) as "transform_string_to_int";
+------------------------+
| transform_string_to_int |
+------------------------+
|                    254 |
+------------------------+
```

Keywords

```
conv
```

DEGREES

Description

```
degrees(double a)
```

- Function: convert radians into angles
- Return type: double type

Example

```
mysql> select degrees(pi());
+--------------+
| degrees(pi()) |
+--------------+
|          180 |
+--------------+
```

Keywords

```
degrees
```

E

Description

```
e()
```

- Function: return the mathematical constant e

- Return type: double type

Example

```
mysql> select e();
+------------------+
| e()              |
+------------------+
| 2.7182818284590451 |
+------------------+
```

Keywords

```
e
```

EXP,DEXP

Description

```
exp(double a)
dexp(double a)
```

- Function: return the a power of e
- Return type: double type

Example

```
mysql> select exp(2);
+-----------------+
| exp(2.0)        |
+-----------------+
| 7.38905609893065 |
+-----------------+

mysql> select dexp(2);
+-----------------+
| dexp(2.0)       |
+-----------------+
| 7.38905609893065 |
+-----------------+
```

Keywords

```
exp, dexp
```

MOD

Description

```
mod(numeric_type a, same_type b)
```

- Function: return the remainder of a divided by b, equivalent to % arithmetic operator.
- Return type: same with the input type

Example

```
 mysql> select mod(10,3);
+-----------+
| mod(10, 3) |
+-----------+
|          1 |
+-----------+

mysql> select mod(5.5,2);
+------------+
| mod(5.5, 2) |
+------------+
|         1.5 |
+------------+
```

Keywords

```
 mod
```

FMOD

Description

```
 fmod(double a, double b)
fmod(float a, float b)
```

- Function: return the remainder of a divided by b, which is equivalent to % arithmetic operator
- Return type: float or double type

Example

```
 mysql> select fmod(10,3);
+-----------------+
| fmod(10.0, 3.0) |
+-----------------+
|               1 |
+-----------------+

mysql> select fmod(5.5,2);
+----------------+
| fmod(5.5, 2.0) |
+----------------+
|            1.5 |
+----------------+
```

Keywords

```
fmod
```

PMOD

Description

```
pmod(int a, int b)
pmod(double a, double b)
```

- Function: positive complementary function
- Return type: int type or double type (depending on input parameters)

Example

```
mysql> select pmod(3, 2), pmod(1.1, 2);
+-----------+-------------+
| pmod(3, 2) | pmod(1, 2) |
+-----------+-------------+
|          1 |           1 |
+-----------+-------------+
```

Keywords

```
pmod
```

GREATEST

Description

```
 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 ...])
```

- Function: return the maximum value in the list
- Return type: same with the parameter type

Example

```
 mysql> select greatest(1,2,3);
+----------------+
| greatest(1, 2, 3) |
+----------------+
|              3 |
+----------------+

mysql> select greatest("a", "b", "c");
+------------------------+
| greatest('a', 'b', 'c') |
+------------------------+
| c                      |
+------------------------+
```

Keywords

```
 greatest
```

LEAST

Description

```
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 ...])
```

- Function: return the minimum value in the list
- Return type: same with the parameter type

Example

```
mysql> select least(1,2,3);
+--------------+
| least(1, 2, 3) |
+--------------+
|            1 |
+--------------+

mysql> select least("a", "b", "c");
+--------------------+
| least('a', 'b', 'c') |
+--------------------+
| a                  |
+--------------------+
```

Keywords

```
least
```

HEX

Description

```
hex(bigint a)
hex(string a)
```

- Function: return the hexadecimal representation of an integer or each character in a string.
- Return type: string type

Example

```
mysql> select hex('abc');
+------------+
| hex('abc') |
+------------+
| 616263     |
+------------+

mysql> select unhex(616263);
+--------------+
| unhex(616263) |
+--------------+
| abc           |
+--------------+
```

Keywords

```
hex
```

UNHEX

Description

```
unhex(string a)
```

- Function: convert a string in hexadecimal format to its original format
- Return type: string type

Example

```
mysql> select hex('abc');
+------------+
| hex('abc') |
+------------+
| 616263     |
+------------+

mysql> select unhex(616263);
+--------------+
| unhex(616263) |
+--------------+
| abc           |
+--------------+
```

Keywords

```
unhex
```

LN

Description

```
ln(double a)
```

- Function: return the natural logarithm of 2
- Return type: double type

Example

```
mysql> select ln(2);
+--------------------+
| ln(2.0)            |
+--------------------+
| 0.69314718055994529 |
+--------------------+
```

Keywords

```
ln
```

DLOG1

Description

```
dlog1(double a)
```

- Function: return the natural logarithmic form of the parameter
- Return type: double type

Example

```
mysql> select dlog1(2);
+--------------------+
| dlog1(2.0)         |
+--------------------+
| 0.69314718055994529 |
+--------------------+
```

Keywords

```
dlog1
```

LOG,LOG10,DLOG10,LOG2

Description

```
log(double base, double a)
```

- Function: return the logarithmic value of log with base as base number and a as exponent.
- Return type: double type

```
log10(double a)
dlog10(double a)
```

- Function: return the logarithmic value of log with 10 as base number and a as exponent.
- Return type: double type

```
log2(double a)
```

- Function: return the logarithmic value of log with 2 as base number and a as exponent.
- Return type: double type

Example

```
mysql> select log(2, 65536);
+------------------+
| log(2.0, 65536.0) |
+------------------+
|               16 |
+------------------+

mysql> select log10(2);
+-------------------+
| log10(2.0)        |
+-------------------+
| 0.3010299956639812 |
+-------------------+

mysql> select dlog10(2);
+-------------------+
| dlog10(2.0)       |
+-------------------+
| 0.3010299956639812 |
+-------------------+

mysql> select log2(2);
+----------+
| log2(2.0) |
+----------+
|        1 |
+----------+
```

Keywords

```
log, log10, dlog, log2
```

NEGATIVE

Description

```
negative(int a)
negative(double a)
```

- Function: take the negative sign bit of parameter a, and return a positive value if the parameter is negative value
- Return type: return the int type or double type according to the input parameter type
- Instructions for use: if you need to ensure that all return values are negative, you can use `-abs(a)` function.

Example

```
mysql> select negative(1.0);
+--------------+
| negative(1.0) |
+--------------+
|           -1 |
+--------------+
1 row in set (0.02 sec)

mysql> select negative(-1);
+--------------+
| negative(-1) |
+--------------+
|            1 |
+--------------+
```

Keywords

```
negative
```

POSITIVE

Description

```
positive(int a)
```

- Function: return the original value of parameters, even if the parameter is negative.
- Return type: int type
- Instructions for use: If you need to ensure that all return values are positive, you can use the`abs()` function.

Example

```
mysql> select positive(-1), positive(1);
+-------------+------------+
| positive(-1) | positive(1) |
+-------------+------------+
|          -1 |          1 |
+-------------+------------+
```

Keywords

```
positive
```

PI

Description

```
pi()
```

- Function: return the constant Pi
- Return type: double type

Example

```
mysql> select pi();
+------------------+
| pi()             |
+------------------+
| 3.1415926535897931 |
+------------------+
```

Keywords

```
pi
```

POW,POWER,DPOW,FPOW

Description

```
pow(double a, double p)
power(double a, double p)
dpow(double a, double p)
fpow(double a, double p)
```

- Function: return the p power of a
- Return type: double type

Example

```
mysql> select pow(2, 10), power(2, 10), dpow(2, 10), fpow(2, 10);
+-------------+---------------+---------------+---------------+
| pow(2.0, 10.0) | power(2.0, 10.0) | dpow(2.0, 10.0) | fpow(2.0, 10.0) |
+-------------+---------------+---------------+---------------+
|          1024 |          1024 |          1024 |          1024 |
+-------------+---------------+---------------+---------------+
```

Keywords

```
POW, POWER, DPOW, FPOW
```

RADIANS

Description

```
radians(double a)
```

- Function: convert radians into angles
- Return type: double type

Example

```
mysql> select radians(90);
+------------------+
| radians(90.0)    |
+------------------+
| 1.5707963267948966 |
+------------------+
```

Keywords

```
radians
```

RAND,RANDOM

Description

```
 rand()
random()
```

- Function: return a random value from 0 to 1.
- Return type: double

Example

```
 mysql> select rand(), rand(), random();
+------------------+------------------+------------------+
| rand()           | rand()           | random()         |
+------------------+------------------+------------------+
| 0.39794450929180808 | 0.34321919244300736 | 0.38788449829415106 |
+------------------+------------------+------------------+
```

Keywords

```
 rand, random
```

ROUND

Description

```
 round(double a)
round(double a, int d)
```

- Function: it is a function for rounding. If it contains only one parameter, the function returns the nearest integer to the value. If it contains 2 parameters, the second parameter is the number of digits retained after the decimal point.

- Return type: if the parameter is floating point type, then the function will return bigint. If the second parameter is greater than 1, then the function will return double type.

Example

```
mysql> select round(100.456, 2);
+------------------+
| round(100.456, 2) |
+------------------+
|           100.46 |
+------------------+
```

Keywords

```
round
```

SIGN

Description

```
sign(double a)
```

- Function: If a is an integer or 0, it will return 1; If a is negative, it will return -1
- Return type: int type

Example

```
mysql> select sign(-1), sign(1.2);
+-----------+----------+
| sign(-1.0) | sign(1.2) |
+-----------+----------+
|         -1 |         1 |
+-----------+----------+
```

Keywords

```
sign
```

SQRT,DSQRT

Description

```
sqrt(double a)
dsqrt(double a)
```

- Function: return the square root of a
- Return type: double type

Example

```
mysql> select sqrt(4), dsqrt(10);
+----------+-------------------+
| sqrt(4.0) | dsqrt(10.0)       |
+----------+-------------------+
|         2 | 3.1622776601683795 |
+----------+-------------------+
```

Keywords

```
sqrt, dsqrt
```

TRUNCATE

Description

```
truncate(double num, int len)
```

- Function: intercept num and retain the specified decimal places of len
- Return type: double type

Example

```
select truncate(1.1234, 2);
+--------------------+
| truncate(1.1234, 2) |
+--------------------+
|               1.12 |
+--------------------+
```

Keywords

```
truncate
```

Syntactical Help

DDL

CREATE-FILE

CREATE FILE

Description

This statement is used to create and upload a file to Palo cluster.

This function is often used to manage files that are required in other commands, such as certificates, public keys and private keys, etc.

```
CREATE FILE "file_name" [IN database]
[properties]
```

- `file_name` : custom file name.
- `database` : the file belongs to a certain database. If the database is not specified, then use the current database.
- `properties` supports the following parameters:
  - `url` : required. The download path of files must be specified. Currently, it only supports unauthenticated http download paths. After the command is successfully executed, the file will be saved in Palo, and the url will no longer be needed.
  - `catalog` : required. The classification name of the file can be customized. But in some commands, it will need to search the files in specified catalog. For example, in routine import, when the data source is kafka, it will need to search files in the catalog of kafka.
  - `md5`: optional. md5 of files. If specified, it will be verified after the file is downloaded.

Example

1. Create file ca.pem and classify it as kafka

```
CREATE FILE "ca.pem"
PROPERTIES
(
    "url" = "https://test.bj.bcebos.com/kafka-key/ca.pem",
    "catalog" = "kafka"
);
```

2. Create file client.key and classify as itmy_catalog

```
CREATE FILE "client.key"
IN my_database
PROPERTIES
(
    "url" = "https://test.bj.bcebos.com/kafka-key/client.key",
    "catalog" = "my_catalog",
    "md5" = "b5bb901bf10f99205b39a46ac3557dd9"
);
```

Keywords

```
CREATE, FILE
```

Best Practices

1. Only users with amdin privilege can execute this command. For files belonging to a database, they can be used by users who have access to the database.
2. Limits on file size and quantity.

   This function is mainly used to manage small files such as certificates. Therefore, the size of a single file is limited to 1MB. A Palo cluster can upload up to 100 files.

CREATE-TABLE

CREATE TABLE

Description

This command is used to create a table. This article mainly introduces the syntax of creating tables self-maintained by Palo. For syntax of external tables, refer to [CREATE EXTERNAL TABLE](PALO/Operating Manual/Data Load /Synchronize Data Through External Table.md#) Document.

```
CREATE TABLE [IF NOT EXISTS] [database.]table
(
    column_definition_list,
    [index_definition_list]
)
[engine_type]
[keys_type]
[table_comment]
[partition_info]
distribution_info
[rollup_list]
[properties]
[extra_properties]
```

- `column_definition_list`

  Column definition list:

  `column_definition[, column_definition]`

  - `column_definition`

    Column definition:

    `column_name column_type [KEY] [aggr_type] [NULL] [default_value] [column_comment]`

    - `column_type`

      Column type, the following types are supported:

```
 TINYINT（1 byte）
  Range: -2^7 + 1 ~ 2^7 - 1
SMALLINT（2 bytes）
   Range: -2^15 + 1 ~ 2^15 - 1
INT（4 bytes）
   Range: -2^31 + 1 ~ 2^31 - 1
BIGINT（8 bytes）
  Range: -2^63 + 1 ~ 2^63 - 1
LARGEINT（16 bytes）
   Range: -2^127 + 1 ~ 2^127 - 1
FLOAT（4 bytes）
   support scientific counting method
DOUBLE（12 bytes）
   support scientific counting method
DECIMAL[(precision, scale)] (16 bytes)
   guarantee the decimal type of precision, default to DECIMAL(10, 0)
   precision: 1 ~ 27
   scale: 0 ~ 9
   In which, the integer part is 1 ~ 18
   not support the scientific counting method
DATE（3 bytes）
   Range: 0000-01-01 ~ 9999-12-31
DATETIME（8 bytes）
   Range:0000-01-01 00:00:00 ~ 9999-12-31 23:59:59
CHAR[(length)]
   fixed-length string. Range of length: 1 ~ 255. Default to 1
VARCHAR[(length)]
   Variable-length string. Range of length: 1 ~ 65533. Default to 1
HLL (1~16385 bytes)
   HyperLogLog column type,there is no need to specify the length and default value.The length is
controlled in the system according to the aggregation degree of data.
  Must be used in combination with HLL_UNION aggregation type.
BITMAP
   bitmap column type,there is no need to specify the length and default value.It represents the sets of
integers, elements support up to 2^64 - 1.
   Must be used in combination with BITMAP_UNION aggregation type.
```

- aggr_type

  Aggregation type, the following types are supported:

```
 SUM：sum.Applicable to numeric type.
MIN：minimum value. Applicable to numeric type.
MAX：maximum value. Applicable to numeric type.
REPLACE：replace. For rows with the same dimension column, the index columns will be imported in the
order of import, and replace the one imported previously with the one imported later.
REPLACE_IF_NOT_NULL：replace if not null. The difference between it with REPLACE lies in only replacing
values not null.
HLL_UNION：aggregation method for columns of HLL type, aggregate by HyperLogLog algorithm.
BITMAP_UNION：aggregation method for columns of BIMTAP type, aggregate the union set of bitmaps.
```

Example:

```
 ```
k1 TINYINT,
k2 DECIMAL(10,2) DEFAULT "10.5",
k4 BIGINT NULL DEFAULT VALUE "1000" COMMENT "This is column k4",
v1 VARCHAR(10) REPLACE NOT NULL,
v2 BITMAP BITMAP_UNION,
v3 HLL HLL_UNION,
v4 INT SUM NOT NULL DEFAULT "1" COMMENT "This is column v4"
```
```

- index_definition_list

  Index list definition:

  index_definition[, index_definition]

  - index_definition

    Index definition:

```
INDEX index_name (col_name[, col_name, ...]) [USING BITMAP] COMMENT 'xxxxxx'
```

    Example:

```
INDEX idx1 (k1, k2) USING BITMAP COMMENT "This is a bitmap index"
```

- engine_type

  Table engine type. All types in this document are OLAP. For other external table engine types, refer to CREATE EXTERNAL TABLE document. Example:

  ENGINE=olap

- key_desc

  Data model.

  key_type(col1, col2, ...)

  key_type supports the following models:

  - DUPLICATE KEY（default）: columns specified after that are ordering columns.
  - AGGREGATE KEY : the columns specified after that are dimensional columns.

- UNIQUE KEY : the columns specified after that are key columns.

Example:

```
 DUPLICATE KEY(col1, col2),
AGGREGATE KEY(k1, k2, k3),
UNIQUE KEY(k1, k2)
```

- table_comment

  Table comment. Example:

  ```
  COMMENT "This is my first PALO table"
  ```

- partition_desc

  Partition information, support two kinds of writing:

  1. LESS THAN : only define the upper bound of partition. Lower bound is determined by the upper bound of last partition.

     ```
      PARTITION BY RANGE(col1[, col2, ...])
     (
        PARTITION partition_name1 VALUES LESS THAN MAXVALUE|("value1", "value2", ...),
        PARTITION partition_name2 VALUES LESS THAN MAXVALUE|("value1", "value2", ...)
     )
     ```

  2. FIXED RANGE: define the left-closed and right open interval of partition.

     ```
      PARTITION BY RANGE(col1[, col2, ...])
     (
        PARTITION partition_name1 VALUES [("k1-lower1", "k2-lower1", "k3-lower1",...), ("k1-upper1", "k2-upper1",
     "k3-upper1", ...)),
        PARTITION partition_name2 VALUES [("k1-lower1-2", "k2-lower1-2", ...), ("k1-upper1-2", MAXVALUE, ))
     )
     ```

- distribution_desc

  Define the data bucketing method.

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

- rollup_list

  Multiple materialized views (ROLLUP) can be created while creating tables.

  ROLLUP (rollup_definition[, rollup_definition, ...])

  - rollup_definition

    rollup_name (col1[, col2, ...]) [DUPLICATE KEY(col1[, col2, ...])] [PROPERTIES("key" = "value")]

    Example:

    ```
     ROLLUP (
       r1 (k1, k3, v1, v2),
       r2 (k1, v1)
     )
    ```

- properties

  Set table properties. Currently the following properties are supported:

  - replication_num

Replication number. The default replication number is 3. If the number of BE node is less than 3, it is required to specify the replication number less than or equal to the number of BE node.

- storage_medium/storage_cooldown_time

  Data storage medium.storage_medium is used to declare the initial storage medium of table data, while storage_cooldown_time is used to set the expiration time. Example:

  ```
    "storage_medium" = "SSD",
  "storage_cooldown_time" = "2020-11-20 00:00:00"
  ```

  This example indicates that the data is stored in SSD and will be automatically migrated to HDD storage after it expires at 00: 00: 00 on 2020-11-20.

- colocate_with

  When it needs to use Colocation Join function, use this parameter to set Colocation Group.

  "colocate_with" = "group1"

- bloom_filter_columns

  Users specify the list of column names to be added with Bloom Filter index. The Bloom Filter index of each column is independent, not combined.

  "bloom_filter_columns" = "k1, k2, k3"

- in_memory

  Set whether the table is Memory Table through this property.

  "in_memory" = "true"

- function_column.sequence_type

  When using the UNIQUE KEY model, a sequence column can be specified. When the KEY columns are the same, REPLACE will be done by the sequence column (use the larger value to replace the smaller value, otherwise the replacement cannot be executed)

  Here we only need to specify the type of sequence columns, time type or integer type is supported. Palo will create a hidden sequence column.

  "function_column.sequence_type" = 'Date'

- Dynamic partition related

  Parameters related to dynamic partition are as follows:

  - dynamic_partition.enable: used to specify whether the dynamic partitioning function at table level is turned on. Default to true。
  - dynamic_partition.time_unit: Used to specify the time unit for dynamically adding partitions, which can be selected as DAY, WEEK and MONTH
  - dynamic_partition.start: Used to specify how many partitions are deleted forward. The value must be less than 0. Default to Integer.MIN_VALUE。
  - dynamic_partition.end: Used to specify the number of partitions created in advance. The value must be greater than 0.
  - dynamic_partition.prefix: Used to specify the prefix of the created partition name. for example, if the prefix of the partition name is p, the partition name will be automatically created as p20200108
  - dynamic_partition.buckets: Used to specify the number of partition and buckets automatically created

Example

1. Create a detailed model table

```
CREATE TABLE example_db.table_hash
(
    k1 TINYINT,
    k2 DECIMAL(10, 2) DEFAULT "10.5",
    k3 CHAR(10) COMMENT "string column",
    k4 INT NOT NULL DEFAULT "1" COMMENT "int column"
)
COMMENT "my first table"
DISTRIBUTED BY HASH(k1) BUCKETS 32
```

2. Create a detailed model table, partition and specify the ordering column, set replication number as 1

```
CREATE TABLE example_db.table_hash
(
    k1 DATE,
    k2 DECIMAL(10, 2) DEFAULT "10.5",
    k3 CHAR(10) COMMENT "string column",
    k4 INT NOT NULL DEFAULT "1" COMMENT "int column"
)
DUPLICATE KEY(k1, k2)
COMMENT "my first table"
PARTITION BY RANGE(k1)
(
    PARTITION p1 VALUES LESS THAN ("2020-02-01"),
    PARTITION p1 VALUES LESS THAN ("2020-03-01"),
    PARTITION p1 VALUES LESS THAN ("2020-04-01")
)
DISTRIBUTED BY HASH(k1) BUCKETS 32
PROPERTIES (
    "replication_num" = "1"
);
```

3. Create a unique key model table, set the initial storage medium and cooldown time

```
CREATE TABLE example_db.table_hash
(
    k1 BIGINT,
    k2 LARGEINT,
    v1 VARCHAR(2048) REPLACE,
    v2 SMALLINT SUM DEFAULT "10"
)
UNIQUE KEY(k1, k2)
DISTRIBUTED BY HASH (k1, k2) BUCKETS 32
PROPERTIES(
    "storage_medium" = "SSD",
    "storage_cooldown_time" = "2015-06-04 00:00:00"
);
```

4. Create an aggregate model table and describe by partition with fixed zone

```
CREATE TABLE table_range
(
    k1 DATE,
    k2 INT,
    k3 SMALLINT,
    v1 VARCHAR(2048) REPLACE,
    v2 INT SUM DEFAULT "1"
)
AGGREGATE KEY(k1, k2, k3)
PARTITION BY RANGE (k1, k2, k3)
(
    PARTITION p1 VALUES [("2014-01-01", "10", "200"), ("2014-01-01", "20", "300")),
    PARTITION p2 VALUES [("2014-06-01", "100", "200"), ("2014-07-01", "100", "300"))
)
DISTRIBUTED BY HASH(k2) BUCKETS 32
```

5. Create an aggregate model table containing HLL and BITMAP column types

```
CREATE TABLE example_db.example_table
(
    k1 TINYINT,
    k2 DECIMAL(10, 2) DEFAULT "10.5",
    v1 HLL HLL_UNION,
    v2 BITMAP BITMAP_UNION
)
ENGINE=olap
AGGREGATE KEY(k1, k2)
DISTRIBUTED BY HASH(k1) BUCKETS 32
```

6. Create two tables self-maintained by same Colocation Group.

```
CREATE TABLE t1 (
    id int(11) COMMENT "",
    value varchar(8) COMMENT ""
)
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 10
PROPERTIES (
    "colocate_with" = "group1"
);

CREATE TABLE t2 (
    id int(11) COMMENT "",
    value1 varchar(8) COMMENT "",
    value2 varchar(8) COMMENT ""
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 10
PROPERTIES (
    "colocate_with" = "group1"
);
```

7. Create a memory table with bitmap index and bloom filter index

```
CREATE TABLE example_db.table_hash
(
    k1 TINYINT,
    k2 DECIMAL(10, 2) DEFAULT "10.5",
    v1 CHAR(10) REPLACE,
    v2 INT SUM,
    INDEX k1_idx (k1) USING BITMAP COMMENT 'my first index'
)
AGGREGATE KEY(k1, k2)
DISTRIBUTED BY HASH(k1) BUCKETS 32
PROPERTIES (
    "bloom_filter_columns" = "k2",
    "in_memory" = "true"
);
```

8. Create a dynamic partition table.

   Create partitions for three days in the table in advance every day, and delete partitions three days before. For example, today is `2020-01-08`, then the partitions named as `p20200108`, `p20200109`, `p20200110`, `p20200111` will be created. The ranges of such partitions are respectively:

```
 [types: [DATE]; keys: [2020-01-08]; ··types: [DATE]; keys: [2020-01-09]; )
[types: [DATE]; keys: [2020-01-09]; ··types: [DATE]; keys: [2020-01-10]; )
[types: [DATE]; keys: [2020-01-10]; ··types: [DATE]; keys: [2020-01-11]; )
[types: [DATE]; keys: [2020-01-11]; ··types: [DATE]; keys: [2020-01-12]; )
```

```
 CREATE TABLE example_db.dynamic_partition
(
    k1 DATE,
    k2 INT,
    k3 SMALLINT,
    v1 VARCHAR(2048),
    v2 DATETIME DEFAULT "2014-02-04 15:36:00"
)
DUPLICATE KEY(k1, k2, k3)
PARTITION BY RANGE (k1) ()
DISTRIBUTED BY HASH(k2) BUCKETS 32
PROPERTIES(
    "dynamic_partition.time_unit" = "DAY",
    "dynamic_partition.start" = "-3",
    "dynamic_partition.end" = "3",
    "dynamic_partition.prefix" = "p",
    "dynamic_partition.buckets" = "32"
);
```

9. Create a table with materialized view (ROLLUP).

```
 CREATE TABLE example_db.rolup_index_table
(
    event_day DATE,
    siteid INT DEFAULT '10',
    citycode SMALLINT,
    username VARCHAR(32) DEFAULT '',
    pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY(event_day, siteid, citycode, username)
DISTRIBUTED BY HASH(siteid) BUCKETS 10
ROLLUP (
    r1(event_day,siteid),
    r2(event_day,citycode),
    r3(event_day)
)
PROPERTIES("replication_num" = "3");
```

Keyword

```
CREATE, TABLE
```

Best Practices

Partition and bucketing

In a table, a bucketing column must be specified, and a partition may not. Detailed introduction of partition and bucketing can be found in Data Division document.

Tables in Palo can be divided into partitioned tables and non-partitioned tables. This property is determined when the table is created, and cannot be changed afterwards. For partitioned tables, operations such as addition, deletion can be done during the process of use, while for non-partitioned tables, such operations are not allowed.

At the same time, the partition columns and bucketing columns cannot be changed after the table is created, neither the type of partition column and bucketing column can be changed, nor any addition or deletion operation can be performed on these columns.

So it is suggested to confirm the usage mode before building the table.

Dynamic Partition

The dynamic partition function is mainly used to help users manage partitions automatically. After set with related rules, Palo system regularly adds new partitions or deletes historical partitions. For more help, refer to Dynamic Partition document.

Materialized Views

Users can create multiple materialized views (ROLLUP) while building tables. Materialized views can also be added after the table is created. It can facilitate users to create all materialized views at one time by writing it in table building statements.

If materialized views are created during table building process, all subsequent data import operations will generate materialized view data synchronously. The number of materialized views may affect the efficiency of data import.

If materialized views are added during later use, if there is data in the table, the creation time of materialized views depends on the current data amount.

Introduction to materialized views can be found in Materialized Views.

Index

Users can create indexes of multiple columns while building tables. Indexes can also be added after the table is created.

When adding index in the later use, if there is data existing in the table, it is required to rewrite all the data, and the index creation time will be determined by the current data volume.

Memory Table

When the `"in_memory" = "true"` property is specified when building table, Palo will cache the data blocks of this table in the PageCache of the storage engine as far as possible to reduce the disk IO. However, this property cannot guarantee the data block is stored in the memory all the time but only works as an indicator of best-effort.

CREATE-REPOSITORY

CREATE REPOSITORY Description

This statement is used to create a repository. Repository is used for backup or recovery. Only admin users can create repository.

```
 CREATE [READ ONLY] REPOSITORY `repo_name`
WITH BROKER `bos`
ON LOCATION `repo_location`
PROPERTIES ("key"="value", ...);
```

- READ ONLY

  If READ ONLY is specified, the repository is read-only, that is, users can only operate recovery and cannot operate backup.

- WITH BROKER

  The create operation depends on the Broker service process. After that, the PROPERTIES should be filled with information for Broker to access the remote repository.

- ON LOCATION

  Specify the path for data storage in remote storage.

Example

1. Create a repository named bos_repo , the data root directory is:bos://my_bucket/palo_backup/

```
 CREATE REPOSITORY `bos_repo`
WITH BROKER `bos_broker`
ON LOCATION "bos://my_bucket/palo_backup/"
PROPERTIES
(
    "bos_endpoint" = "http://gz.bcebos.com",
    "bos_accesskey" = "069fc2786e664e63a5f111111114ddbs22",
    "bos_secret_accesskey"="70999999999999de274d59eaa980a"
);
```

Keywords

```
CREATE, REPOSITORY
```

Best Practices

1. Multiple repositories can be created in a cluster. Only users with ADMIN privileges can create repository.
2. Any user can view the repository(ies) created by the SHOW REPOSITORIES command.
3. When operating for data migration, it is necessary to create the same repository in the source cluster and the target cluster, so that the data snapshot backed up by the source cluster can be viewed in target cluster through this repository.

CREATE-ODBC-TABLE

CREATE ODBC TABLE

Description

Palo supports the reading and writing operations by creating external tables and accessing external data sources through ODBC protocol. Currently the ODBC data sources supported on the cloud include:

- MySQL
- Oracle
- PostgreSQL

When creating the ODBC external table, it is required to fill in the external data source connection information in the table building statement.

There are two ways to provide connection information.

One is to directly connect the information to describe the table building statement.

One is to manage the connection information in a unified way by creating resources, and cite the resources in the table building statement.

It is recommended to use the second method to manage the connection information. Creating resources can be found in Resource Management file.

```
CREATE EXTERNAL TABLE [db.]tbl_name
(column_definition[, column_definition, ...])
ENGINE=ODBC
[tbl_comment]
[properties]
```

- `[db.]tbl_name`

  Library name and table name. The names may not be consistent with external data sources, and can be customized. There will be a mapping relationship in properties.

- `column_definition`

  Column definition. The name of the column should be consistent with the name in the external data source. The number and order of columns may not be the same. Whether the column is allowed to be NULL should be consistent with that in the source database, otherwise problems may occur in query or write.

  `col_name DOUBLE NULL COMMENT "abc"` `col_name VARCHAR(1) NOT NULL COMMENT "def"`

- `tbl_comment`

  Table comment

- `properties`

  External data source information.

  1. Define external data sources by referencing resources.

```
PROPERTIES (
  "odbc_catalog_resource" = "resource_name",
  "database" = "oracle_db",
  "table" = "oracle_tbl"
);
```

- odbc_catalog_resource : specify the name of resource.
- database : name of the database in the external data source.
- table : name of table in the external data source.

2. Define external data sources directly.

```
PROPERTIES (
  "host" = "192.168.0.1",
  "port" = "8086",
  "user" = "test",
  "password" = "test",
  "database" = "test",
  "table" = "baseall",
  "driver" = "oracle",
  "odbc_type" = "oracle"
);
```

- host/port : ODBC protocol connection target of external data source.
- user/passwd : User name and password for accessing external data source.
- database/table : name of the corresponding database and table in the external data source.
- driver : Name of ODBC Driver. Public cloud users may choose:PostgreSQL, MySQL,Oracle,SQLServer.
- odbc_type: data source type, supporting:oracle, mysql, postgresql.

Example

1. Create a MySQL external table. Use the ODBC Resource created.

```
CREATE EXTERNAL TABLE `baseall_oracle` (
  `k1` decimal(9, 3) NOT NULL COMMENT "",
  `k2` char(10) NOT NULL COMMENT "",
  `k3` datetime NOT NULL COMMENT "",
  `k5` varchar(20) NOT NULL COMMENT "",
  `k6` double NOT NULL COMMENT ""
) ENGINE=ODBC
COMMENT "ODBC"
PROPERTIES (
"odbc_catalog_resource" = "mysql_odbc",
"database" = "test",
"table" = "baseall"
);
```

2. Create an Oracle external table. Set the connection mode directly.

```
CREATE EXTERNAL TABLE `baseall_oracle` (
  `k1` decimal(9, 3) NOT NULL COMMENT "",
  `k2` char(10) NOT NULL COMMENT "",
  `k3` datetime NOT NULL COMMENT "",
  `k5` varchar(20) NOT NULL COMMENT "",
  `k6` double NOT NULL COMMENT ""
) ENGINE=ODBC
COMMENT "ODBC"
PROPERTIES (
    "host" = "192.168.0.1",
    "port" = "8086",
    "user" = "test",
    "password" = "test",
    "database" = "test",
    "table" = "baseall",
    "driver" = "Oracle",
    "odbc_type" = "oracle"
);
```

Keywords

```
CREATE, MYSQL, ORACLE, ODBC, EXTERNAL, TABLE
```

Best Practices

1. Query ODBC external tables

   The query of ODBC external table is the same as the query of ordinary table, so the query can be done by using SQL statement.

   In essence, the query of external tables in Palo is to connect and query external data sources through ODBC client on a Compute Node node. Therefore, the external data source and the Compute Node network should be connected in both directions.

   At the same time, The query of external tables in Palo is not in distributed way but by connecting single Client. Therefore, its performance efficiency is far lower than that of querying the table stored in Palo. External tables are more applicable to the correlation query of some frequently updated dimension tables and fact tables stored in Palo, or synchronization of data from an external data source to Palo by `INSERT INTO SELECT` .

2. Write into external table

   Data can be directly written into ODBC external tables by `INSERT` command. Specific operations can be found in: Export Data to External Tables.

3. Corresponding relations between column types

   Different data sources have different column types. Here are the mapping relationships between the column types of three ODBC data sources and Palo column types.

   1. MySQL

MySQL	Doris	Alternative Plan
BOOLEAN	BOOLEAN	
CHAR	CHAR	Currently support UTF8 encoding only
VARCHAR	VARCHAR	Currently support UTF8 encoding only
DATE	DATE	
FLOAT	FLOAT	
TINYINT	TINYINT	
SMALLINT	SMALLINT	
INT	INT	
BIGINT	BIGINT	
DOUBLE	DOUBLE	
DATETIME	DATETIME	
DECIMAL	DECIMAL	

   2. PostgreSQL

PostgreSQL	Doris	Alternative Plan
BOOLEAN	BOOLEAN	
CHAR	CHAR	Currently support UTF8 encoding only
VARCHAR	VARCHAR	Currently support UTF8 encoding only
DATE	DATE	
REAL	FLOAT	
SMALLINT	SMALLINT	
INT	INT	
BIGINT	BIGINT	
DOUBLE	DOUBLE	
TIMESTAMP	DATETIME	
DECIMAL	DECIMAL	

3. Oracle

Oracle	Doris	Alternative Plan
Not Support	BOOLEAN	Oracle can use number(1) to replace boolean
CHAR	CHAR	
VARCHAR	VARCHAR	
DATE	DATE	
FLOAT	FLOAT	
None	TINYINT	Oracle can be replaced with NUMBER
SMALLINT	SMALLINT	
INT	INT	
None	BIGINT	Oracle can be replaced with NUMBER
None	DOUBLE	Oracle can be replaced with NUMBER
DATETIME	DATETIME	
NUMBER	DECIMAL	

CREATE-MATERIALIZED-VIEW

CREATE MATERIALIZED VIEW

description

This statement is used to create materialized views.

The operation is asynchronous, progress of which can be viewed through SHOW ALTER TABLE MATERIALIZED VIEW after successful submission.

```
CREATE MATERIALIZED VIEW [MV name] as [query]
[PROPERTIES ("key" = "value")]
```

- MV name

  Name of materialized view, required item.

  Materialized view names of the same table cannot be duplicated.

- query

  Query statement used to build materialized view, and the result of query statement is materialized view data. Currently the following formats of query are supported:

```
SELECT select_expr[, select_expr ...]
FROM [Base view name]
GROUP BY column_name[, column_name ...]
ORDER BY column_name[, column_name ...]
```

  - select_expr: materialize all columns in the schema of the view.
    - Only single columns and aggregate columns not containing expression calculation are supported.
    - Currently, aggregate functions only support sum, min and max, and the parameters of aggregate functions can only be single columns not containing expression calculation.
    - It contains at least one single column.
    - All columns involved can only appear once.
  - base view name: original table name of materialized view, required item.
    - It must be a single table and not a subquery
  - group by: grouping column of materialized views, optional item.
    - If not filled in, the data will not be grouped.
  - order by: ordering column of material views, required item.
    - The order of declaration of ordering column must be consistent with that in select_expr.
    - If order by is not declared, then the ordering column will be automatically supplemented according to the rules. If the materialized view is an aggregation type, all grouping columns will be automatically supplemented as ordering columns. If the materialized view is not the aggregation type, then the first 36 bytes will be

automatically supplemented as ordering columns. If the number of ordering columns automatically supplemented is less than 3, the first three will be taken as ordering columns.

- If query contains grouping columns, then the ordering columns must be consistent with grouping columns.

- properties

  Declare some configurations of materialized view, optional.

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

  The following configurations are supported:

  - timeout: timeout for materialized view construction.

Example

Assuming that the Base table structure is:

```
mysql> desc duplicate_table;
+-------+--------+------+------+---------+-------+
| Field | Type   | Null | Key  | Default | Extra |
+-------+--------+------+------+---------+-------+
| k1    | INT    | Yes  | true | N/A     |       |
| k2    | INT    | Yes  | true | N/A     |       |
| k3    | BIGINT | Yes  | true | N/A     |       |
| k4    | BIGINT | Yes  | true | N/A     |       |
+-------+--------+------+------+---------+-------+
```

1. Create a materialized view containing only the columns of the original table (k1, k2)

```
create materialized view k1_k2 as
select k1, k2 from duplicate_table;
```

The schema of materialized view is shown in the following figure. The materialized view only contains two columns of k1 and k2 without any aggregation

```
+---------------+-------+--------+------+------+---------+-------+
| IndexName     | Field | Type   | Null | Key  | Default | Extra |
+---------------+-------+--------+------+------+---------+-------+
| k1_k2         | k1    | INT    | Yes  | true | N/A     |       |
|               | k2    | INT    | Yes  | true | N/A     |       |
+---------------+-------+--------+------+------+---------+-------+
```

2. Create a materialized view with k2 as the ordering column

```
create materialized view k2_order as
select k2, k1 from duplicate_table order by k2;
```

The schema of materialized view is shown in the following figure. Materialized view only contains two columns of k2, k1, in which K2 column is an ordering column without any aggregation.

```
+---------------+-------+--------+------+-------+---------+-------+
| IndexName     | Field | Type   | Null | Key   | Default | Extra |
+---------------+-------+--------+------+-------+---------+-------+
| k2_order      | k2    | INT    | Yes  | true  | N/A     |       |
|               | k1    | INT    | Yes  | false | N/A     | NONE  |
+---------------+-------+--------+------+-------+---------+-------+
```

3. Create a materialized view aggregated with k1, k2 as grouping columns and k3 as SUM

```
create materialized view k1_k2_sumk3 as
select k1, k2, sum(k3) from duplicate_table group by k1, k2;
```

The schema of materialized view is shown in the following figure. The materialized view contains two columns of k1, k2 and sum(k3), in which k1 and k2 are grouping columns, and sum(k3) is the sum value of k3 columns grouped according to k1 and k2.

As there is no ordering column declared in materialized view, and the materialized view contains aggregation data, the grouping columns k1, k2 will be supplemented as ordering columns by default.

```
+----------------+-------+--------+------+-------+---------+-------+
| IndexName      | Field | Type   | Null | Key   | Default | Extra |
+----------------+-------+--------+------+-------+---------+-------+
| k1_k2_sumk3    | k1    | INT    | Yes  | true  | N/A     |       |
|                | k2    | INT    | Yes  | true  | N/A     |       |
|                | k3    | BIGINT | Yes  | false | N/A     | SUM   |
+----------------+-------+--------+------+-------+---------+-------+
```

4. Create a materialized view with duplicate rows removed

```
create materialized view deduplicate as
select k1, k2, k3, k4 from duplicate_table group by k1, k2, k3, k4;
```

The schema of materialized view is shown in the following figure. The materialized view contains k1, k2, k3, k4 columns with no duplicate rows.

```
+---------------+-------+--------+------+-------+---------+-------+
| IndexName     | Field | Type   | Null | Key   | Default | Extra |
+---------------+-------+--------+------+-------+---------+-------+
| deduplicate   | k1    | INT    | Yes  | true  | N/A     |       |
|               | k2    | INT    | Yes  | true  | N/A     |       |
|               | k3    | BIGINT | Yes  | true  | N/A     |       |
|               | k4    | BIGINT | Yes  | true  | N/A     |       |
+---------------+-------+--------+------+-------+---------+-------+
```

5. Create a non-aggregation type materialized view with no ordering columns declared

The schema of all_type_table is as follows:

```
+-------+--------------+------+-------+---------+-------+
| Field | Type         | Null | Key   | Default | Extra |
+-------+--------------+------+-------+---------+-------+
| k1    | TINYINT      | Yes  | true  | N/A     |       |
| k2    | SMALLINT     | Yes  | true  | N/A     |       |
| k3    | INT          | Yes  | true  | N/A     |       |
| k4    | BIGINT       | Yes  | true  | N/A     |       |
| k5    | DECIMAL(9,0) | Yes  | true  | N/A     |       |
| k6    | DOUBLE       | Yes  | false | N/A     | NONE  |
| k7    | VARCHAR(20)  | Yes  | false | N/A     | NONE  |
+-------+--------------+------+-------+---------+-------+
```

When the materialized view contains k3, k4, k5, k6, k7 columns, and does not contain declared ordering columns, then create the statement as follows:

```
create materialized view mv_1 as
select k3, k4, k5, k6, k7 from all_type_table;
```

The ordering columns supplemented in the system are k3, k4, k5 and K5 by default. The sum of bytes of these three column types is 4(INT) + 8(BIGINT) + 16(DECIMAL) = 28 < 36. Therefore, these three columns are supplemented as ordering columns.

The schema of materialized view is as follows, the key field of k3, k4, k5 and K5 columns, the ordering columns, is true. The key field of K6 k6, k7 columns, the non-ordering columns, is false.

```
+---------------+-------+-------------+------+-------+---------+-------+
| IndexName     | Field | Type        | Null | Key   | Default | Extra |
+---------------+-------+-------------+------+-------+---------+-------+
| mv_1          | k3    | INT         | Yes  | true  | N/A     |       |
|               | k4    | BIGINT      | Yes  | true  | N/A     |       |
|               | k5    | DECIMAL(9,0)| Yes  | true  | N/A     |       |
|               | k6    | DOUBLE      | Yes  | false | N/A     | NONE  |
|               | k7    | VARCHAR(20) | Yes  | false | N/A     | NONE  |
+---------------+-------+-------------+------+-------+---------+-------+
```

Keywords

```
CREATE, MATERIALIZED, VIEW
```

CREATE-RESOURCE

CREATE RESOURCE

Description

Used to create a resource. Resources can be cited by other processes.

Currently the following resource types are supported:

1. ODBC

```
CREATE EXTERNAL RESOURCE `resource_name`
[PROPERTIES]
```

- `PROPERTIES` is used to set ODBC connection information.
  - `type` : resource type. Fixed content: `odbc_catalog`.

- `host/port` : target of data source connection.
- `user/password` : username and password for connecting data source.
- `database/table` : Database and table names in the data source.

  When citing resources in CREATE ODBC TABLE, the information here can be covered by specifying the database and table names.
- `odbc_type` : ODBC type, supporting:`oracle`,`mysql` and `postgresql`.
- `driver`: ODBC Driver name. The public cloud users support: `Oracle`,`MySQL` and `PostgreSQL`.

Example

1. Create an ODBC resource for connecting MySQL database.

```
CREATE EXTERNAL RESOURCE `mysql_odbc_resource`
PROPERTIES (
    "type" = "odbc_catalog",
    "host" = "192.168.0.1",
    "port" = "8086",
    "user" = "test",
    "password" = "test",
    "database" = "test",
    "table" = "test",
    "odbc_type" = "mysql",
    "driver" = "MySQL"
);
```

2. Create an ODBC resource for connecting Oracle database.

```
CREATE EXTERNAL RESOURCE `oracle_odbc`
PROPERTIES (
    "type" = "odbc_catalog",
    "host" = "192.168.0.1",
    "port" = "8086",
    "user" = "test",
    "password" = "test",
    "odbc_type" = "oracle",
    "driver" = "Oracle"
);
```

Keywords

```
CREATE, RESOURCE
```

DROP-RESOURCE

DROP RESOURCE

Description

Drop the resource created.

```
DROP RESOURCE 'resource_name';
```

Example

1. Drop a resource.

```
DROP RESOURCE 'odbc_resource';
```

Keywords

```
DROP, RESOURCE
```

DROP-REPOSITORY

DROP REPOSITORY Description

This statement is used to drop a created repository. Only users with root or superuser privilege can drop the repository.

```
DROP REPOSITORY `repo_name`;
```

To drop a repository is to delete the mapping of the repository in Palo only, the repository data will not be deleted. After deleting, the repository can be mapped by specifying same parameters again.

Example

1. Drop the repository `bos_repo` :

```
DROP REPOSITORY `bos_repo`;
```

Keywords

```
DROP, REPOSITORY
```

DML

BROKER-LOAD

BROKER LOAD

Description

This command is mainly used to load data from remote storage (such as BOS and HDFS) through Broker service process.

```
 LOAD LABEL load_label
(
data_desc1[, data_desc2, ...]
)
WITH BROKER broker_name
[broker_properties]
[load_properties];
```

- load_label

  Each load needs to specify a unique Label. The job progress can be viewed through this label later.

  [database.]label_name

- data_desc1

  Used to describe a set of files that need to be loaded.

```
 [MERGE|APPEND|DELETE]
DATA INFILE
(
"file_path1"[, file_path2, ...]
)
[NEGATIVE]
INTO TABLE `table_name`
[PARTITION (p1, p2, ...)]
[COLUMNS TERMINATED BY "column_separator"]
[FORMAT AS "file_type"]
[(column_list)]
[COLUMNS FROM PATH AS (c1, c2, ...)]
[PRECEDING FILTER predicate]
[SET (column_mapping)]
[WHERE predicate]
[DELETE ON expr]
[ORDER BY source_sequence]
```

  - [MERGE|APPEND|DELETE]

    The default value is APPEND, which means that this load is an ordinary append write operation. MERGE and DELETE types are only applicable to Unique Key model tables. The MERGE type needs to be used with the [DELETE ON] statement to label the Delete Flag column. The DELETE type means that all the data loaded this time are deleted data.

  - DATA INFILE

    Specify the file path to be loaded. It can be multiple. Wildcards can be used. The path must eventually match the file, and if it only matches the directory, the load will fail.

  - NEGTIVE

    This keyword is used to indicate that this load is a batch of "negative" loads. This method is only for aggregate data tables with integer SUM aggregate type. In this way, the integer value corresponding to the SUM aggregate column in

the loaded data will be inverted. It is mainly used to offset the wrong data loaded before.

- `PARTITION(p1, p2, ...)`

  User can specify some partitions only loaded with tables. Data not in the partition will be ignored.

- `COLUMNS TERMINATED BY`

  Specify the column separator. Valid only in CSV format. Only single byte separators can be specified.

- `FORMAT AS`

  Specify the file type and support CSV, PARQUET and ORC formats. Default to CSV.

- `column list`

  Used to specify the order of columns in the original file. For a detailed introduction of this part, please refer to Mapping, Transformation and Filtering of Columns document.

  `(k1, k2, tmpk1)`

- `COLUMNS FROM PATH AS`

  Specify the columns extracted from the load file path.

- `PRECEDING FILTER predicate`

  Pre-filtering condition. Data are spliced into original data rows in sequence firstly according to `column list` and `COLUMNS FROM PATH AS` , and then filtered by the pre-filtering conditions. For a detailed introduction of this part, please refer to Mapping, Transformation and Filtering of Columns document.

- `SET (column_mapping)`

  Specify the conversion function for the column.

- `WHERE predicate`

  Filter the loaded data according to the conditions. For a detailed introduction of this part, please refer to Mapping, Transformation and Filtering of Columns document.

- `DELETE ON expr`

  It needs to be used together with MEREGE load mode, only for tables of Unique Key model. And it is used to specify the column and calculation relation of Delete Flag in loaded data.

- `ORDER BY`

  Only tables for the Unique Key model. It is used to specify the column representing Sequence Col in the loaded data. It is mainly used to ensure data order during loading.

- `WITH BROKER broker_name`

  Specify the name of the Broker service to use. In the public cloud Palo. Broker service name is `bos`

- `broker_properties`

  Specify the information required by the broker. This information is often used for remote storage systems that Broker can access. For BOS or HDFS. For specific information, please refer to Broker document.

```
(
  "key1" = "val1",
  "key2" = "val2",
  ...
)
```

- `load_properties`

  Specify parameters related to the load. Currently the following parameters are supported:

- `timeout`

  Load timeout period. Default to 4 hours, in seconds.

- `max_filter_ratio`

  Maximum tolerance of filterable (data nonstandard, etc.) data ratio. Zero tolerance by default. The value range is 0 to 1.

- `exec_mem_limit`

  Load memory limits. Default to 2 GB, in bytes.

- `strict_mode`

  Whether there are strict restrictions on data. Default to false.

- `timezone`

  Specify the time zone of some functions affected by time zone, such as `strftime/alignment_timestamp/from_unixtime` etc., for details, please refer to Time Zone document. If not specified, use the "Asia/Shanghai" time zone.

Example

1. Load a batch of data from BOS

```
LOAD LABEL example_db.label1
(
    DATA INFILE("bos://my_bucket/input/file.txt")
    INTO TABLE `my_table`
    COLUMNS TERMINATED BY ","
)
WITH BROKER bos
(
    "bos_endpoint" = "http://bj.bcebos.com",
    "bos_accesskey" = "xxxxxxxxxxxxxxxxxxxxxxxxxxx",
    "bos_secret_accesskey"="yyyyyyyyyyyyyyyyyyyyyyyyy"
);
```

   Load the file `file.txt`, separated by commas, and load it into the table `my_table`.

2. Load data from BOS and use wildcards to match two batches of files. Load into two tables respectively.

```
 LOAD LABEL example_db.label2
(
    DATA INFILE("bos://my_bucket/input/file-10*")
    INTO TABLE `my_table1`
    PARTITION (p1)
    COLUMNS TERMINATED BY ","
    (k1, tmp_k2, tmp_k3)
    SET (
        k2 = tmp_k2 + 1,
        k3 = tmp_k3 + 1
    )
    DATA INFILE("bos://my_bucket/input/file-20*")
    INTO TABLE `my_table2`
    COLUMNS TERMINATED BY ","
    (k1, k2, k3)
)
WITH BROKER bos
(
    "bos_endpoint" = "http://bj.bcebos.com",
    "bos_accesskey" = "xxxxxxxxxxxxxxxxxxxxxxxxx",
    "bos_secret_accesskey"="yyyyyyyyyyyyyyyyyyyyyyyy"
);
```

Use wildcard matching to load two batches of files `file-10*` and `file-20*`。Load into `my_table1` and `my_table2` , in which, `my_table1` is designated to be loaded to `p1` after adding +1 to values in the second columns and third columns in source files loaded.

3. Load a batch of data from HDFS.

```
 LOAD LABEL example_db.label3
(
    DATA INFILE("hdfs://hdfs_host:hdfs_port/user/palo/data/*/*")
    INTO TABLE `my_table`
    COLUMNS TERMINATED BY "\\x01"
)
WITH BROKER my_hdfs_broker
(
    "username" = "",
    "password" = "",
    "dfs.nameservices" = "my_ha",
    "dfs.ha.namenodes.my_ha" = "my_namenode1, my_namenode2",
    "dfs.namenode.rpc-address.my_ha.my_namenode1" = "nn1_host:rpc_port",
    "dfs.namenode.rpc-address.my_ha.my_namenode2" = "nn2_host:rpc_port",
    "dfs.client.failover.proxy.provider" = "org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
);
```

Specify the default delimiter `\\x01` , and use wildcard * to specify all files in all directories under the `data` directory. Use simple authentication and configure namenode HA at the same time.

4. Load Parquet FORMAT data and specify parquet as format. It is judged by file suffix by default.

```
LOAD LABEL example_db.label4
(
    DATA INFILE("bos://bucket/input/file")
    INTO TABLE `my_table`
    FORMAT AS "parquet"
    (k1, k2, k3)
)
WITH BROKER bos
(
    "bos_endpoint" = "http://bj.bcebos.com",
    "bos_accesskey" = "xxxxxxxxxxxxxxxxxxxxxxxxx",
    "bos_secret_accesskey"="yyyyyyyyyyyyyyyyyyyyyyyy"
);
```

5. Load the data and extract the partition fields in the file path

```
LOAD LABEL example_db.label10
(
    DATA INFILE("hdfs://hdfs_host:hdfs_port/input/city=beijing/*/*")
    INTO TABLE `my_table`
    FORMAT AS "csv"
    (k1, k2, k3)
    COLUMNS FROM PATH AS (city, utc_date)
)
WITH BROKER hdfs
(
    "username"="hdfs_user",
    "password"="hdfs_password"
);
```

Columns in `my_table` are `k1, k2, k3, city, utc_date`.

In which, the catalog `hdfs://hdfs_host:hdfs_port/user/palo/data/input/dir/city=beijing` contains the following files:

```
hdfs://hdfs_host:hdfs_port/input/city=beijing/utc_date=2020-10-01/0000.csv
hdfs://hdfs_host:hdfs_port/input/city=beijing/utc_date=2020-10-02/0000.csv
hdfs://hdfs_host:hdfs_port/input/city=tianji/utc_date=2020-10-03/0000.csv
hdfs://hdfs_host:hdfs_port/input/city=tianji/utc_date=2020-10-04/0000.csv
```

The file only contains three columns of data: `k1, k2, k3` , and the two columns of `city, utc_date` will be extracted from the file path.

6. Filter the data to be loaded.

```
LOAD LABEL example_db.label6
(
    DATA INFILE("bos://bucket/input/file")
    INTO TABLE `my_table`
    (k1, k2, k3)
    PRECEDING FILTER k1 = 1
    SET (
        k2 = k2 + 1
    )
    WHERE k1 > k2
)
WITH BROKER bos
(
    "bos_endpoint" = "http://bj.bcebos.com",
    "bos_accesskey" = "xxxxxxxxxxxxxxxxxxxxxxxxx",
    "bos_secret_accesskey"="yyyyyyyyyyyyyyyyyyyyyyyy"
);
```

Only in the original data, k1 = 1 , and after conversion, rows with k1 > k2 will be loaded.

7. Load data, extract the time partition field in the file path, and the time contains %3A (in hdfs path,':' is not allowed, all':' will be replaced by %3A)

```
LOAD LABEL example_db.label7
(
    DATA INFILE("hdfs://host:port/user/data/*/test.txt")
    INTO TABLE `tbl12`
    COLUMNS TERMINATED BY ","
    (k2,k3)
    COLUMNS FROM PATH AS (data_time)
    SET (
        data_time=str_to_date(data_time, '%Y-%m-%d %H%%3A%i%%3A%s')
    )
)
WITH BROKER hdfs
(
    "username"="user",
    "password"="pass"
);
```

There are the following files under the path:

```
 /user/data/data_time=2020-02-17 00%3A00%3A00/test.txt
/user/data/data_time=2020-02-18 00%3A00%3A00/test.txt
```

The table structure is:

```
 data_time DATETIME,
k2        INT,
k3        INT
```

8. Load a batch of data from HDFS, and specify timeout and filtering ratio. Broker using plaintext my_hdfs_broker. Simple authentication. And delete the columns in the original data that match the columns with v2 greater than 100 in the loaded data, and load the other columns normally

```
 LOAD LABEL example_db.label8
(
    MERGE DATA INFILE("bos://bucket/input/file")
    INTO TABLE `my_table`
    (k1, k2, k3, v2, v1)
    DELETE ON v2 > 100
)
WITH BROKER bos
(
    "bos_endpoint" = "http://bj.bcebos.com",
    "bos_accesskey" = "xxxxxxxxxxxxxxxxxxxxxxxxxxx",
    "bos_secret_accesskey"="yyyyyyyyyyyyyyyyyyyyyyyyy"
)
PROPERTIES
(
    "timeout" = "3600",
    "max_filter_ratio" = "0.1"
);
```

Load using MERGE method.`my_table` must be a Unique Key table. When the value of v2 column in the loaded data is greater than 100, the row is considered as a deleted row.

The timeout of the load task is 3600 seconds, and the allowable error rate is within 10%.

9. Specify the source_sequence column during load to ensure the replacement order in the UNIQUE_KEYS table:

```
 LOAD LABEL example_db.label9
(
   DATA INFILE("bos://bucket/input/file")
   INTO TABLE `my_table`
   COLUMNS TERMINATED BY ","
   (k1,k2,source_sequence,v1,v2)
   ORDER BY source_sequence
)
WITH BROKER bos
(
   "bos_endpoint" = "http://bj.bcebos.com",
   "bos_accesskey" = "xxxxxxxxxxxxxxxxxxxxxxxxxxx",
   "bos_secret_accesskey"="yyyyyyyyyyyyyyyyyyyyyyyyy"
)
```

`my_table` must be an Unqiue Key model table with sequcecol specified. Data will be ordered according to the value of `source_sequence` column in the source data.

Keywords

BROKER, LOAD

Best Practices

1. View load task status

   Broker Load is an asynchronous load process. Successful execution of statements only means successful submission of load tasks, but does not mean successful load of data. Load status should be viewed by SHOW LOAD command.

2. Cancel the load task

   Load tasks that have been submitted but not finished can be cancelled by CANCEL LOAD command. After cancellation, the written data will also be rolled back and will not take effect.

3. Label, load transaction, multi-table atomicity

   All load tasks in Palo are atomic. Moreover, the atomicity can be guaranteed by loading multiple tables in the same load task. At the same time, Palo can also ensure that data load is not lost or duplicated through the mechanism of Label.Specific instructions can be found in Load Transactions and Atomicity document.

4. Column mapping, derived columns and filtering

   Palo can support very rich column conversion and filtering operations in load statements. Most built-in functions and UDFs are supported. For how to use this function correctly, please refer to Mapping, transformation and filtering of columns document.

5. Error data filtering

   Palo's load task can tolerate some malformed data. Tolerance rate can be set by `max_filter_ratio` , which is 0 by default, which means that, when there is an error data, the whole load task will fail. If the user wants to ignore some data rows with problems, the secondary parameter can be set to a value between 0 and 1, and Palo will automatically skip the rows with incorrect data format.

   For some calculation methods of tolerance rate, please refer to Mapping, transformation and filtering of columns document.

6. Strict mode

   `strict_mode` property is used to set whether the load task runs in strict mode. This format will affect the results of column mapping, transformation and filtering. For a detailed description of the strict mode, please refer to [Strict Mode]PALO/Operating Manual/Data Load/Strict Mode.md#) document.

7. Timeout period

   The default timeout for Broker Load is 4 hours, starting from the time when the task is submitted.If it is not completed within the timeout period, the task will fail.

8. Data volume and task number limit

   Broker Load is suitable for loading data within 100GB in one load task. Although theoretically there is no upper limit to the amount of data loaded in an load task. However, submitting too large an load will lead to a long running time, and the cost of retry after failure will also increase.

   At the same time, due to the cluster size, we limit the maximum amount of loaded data to the number of ComputeNode nodes * 3GB. So as to ensure the rational utilization of system resources.If there is a large amount of data to be loaded, it is recommended to submit it in multiple load tasks.

   Palo also limits the number of load tasks running simultaneously in the cluster, usually ranging from 3 to 10. Load jobs submitted afterwards will be queued. The maximum queue length is 100. Subsequent submissions will be rejected directly. Note that the queue time is also calculated into the total job time. If it times out, the job will be cancelled. Therefore, it is suggested to reasonably control the frequency of job submission by monitoring the operation status.

BACKUP

BACKUP Description

This statement is used to back up the data in specified database. This command is an asynchronous operation. After successful submission, progress can be viewed by SHOW BACKUP command.

Backup of OLAP-type tables is only supported.

```
BACKUP SNAPSHOT [db_name.]snapshot_name
TO `repository_name`
ON (
    `table_name` [PARTITION (`p1`, ...)],
    ...
)
PROPERTIES ("key"="value", ...);
```

- `snapshot_name`: give the snapshot of this backup a name.
- `ON`: specify the table or partition to back up.
- `PROPERTIES`: specify some parameters
  - `timeout`: Task timeout period, default to one day, in seconds.

Example

1. Backup the table `example_tbl` under `example_db` to respository `example_repo` :

```
BACKUP SNAPSHOT example_db.snapshot_label1
TO example_repo
ON (example_tbl);
```

2. Back up the p1, p2 partition of table example_tbl under example_db, and set the timeout.

```
BACKUP SNAPSHOT example_db.snapshot_label2
TO example_repo
ON
(
    example_tbl PARTITION (p1,p2),
    example_tbl2
)
PROPERTIES
(
    "timeout" = "7200"
);
```

Keywords

```
BACKUP, SNAPSHOT
```

Best Practices

1. Only one backup operation can be performed under the same database.
2. The backup operation backs up the underlying tables of the specified table or partition and Materialized View. And only one copy is backed up.
3. Efficiency of backup operation

   The efficiency of backup operation depends on the amount of data, the number of Compute Node and the number of files. Every Compute Node where backup data slices are located will participate in the upload stage of backup operation. The more nodes, the higher the efficiency of uploading is.

   The amount of file data only involves the number of slices and the number of files in each shard. If there are too many shards, or there are too many small files in the shards, it may increase the backup operation time.

EXPORT

EXPORT Description

This statement is used to export the data of the specified table to the specified location.

This is an asynchronous operation, which returns if the task is submitted successfully. The progress can be view bySHOW EXPORT command.

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

- table_name

  The table name of the table currently being exported. Only export of Palo local table data is supported.

- partition

  Only some specified partitions of a specified table can be exported.

- export_path

  The exported path must be a directory.

- opt_properties

  Used to specify some export parameters.

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

  The following parameters can be specified:

  - column_separator : Specify the column separator for export, the default is \t. Only single byte is supported.
  - line_delimiter : Specify the line delimiter for export, the default is \t. Only single byte is supported.
  - exec_mem_limit : Export the maximum memory usage of a single BE node, which is 2GB by default, and the unit is bytes.
  - timeout : The timeout of load job is 2 hours by default, and the unit is seconds.
  - tablet_num_per_task : The maximum number of Tablet that can be scanned per subtask.

- WITH BROKER

The export function needs to write data to the remote storage through the Broker process. Here, it is required to define relevant connection information for Broker to use.

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

Example

1. Export all data in testTbl table to hdfs

```
EXPORT TABLE testTbl
TO "bos://my_buckets/export/"
WITH BROKER bos
(
    "bos_endpoint" = "http://bj.bcebos.com",
    "bos_accesskey" = "xxxxxxxxxxxxxxxxxxxxxxxxx",
    "bos_secret_accesskey"="yyyyyyyyyyyyyyyyyyyyyyyy"
);
```

2. Export partitions P1 p1, p2 in testTbl table to hdfs.

```
 EXPORT TABLE testTbl
TO "bos://my_buckets/export/"
(
   "column_separator" = ",",
   "tablet_num_per_task" = "10"
)
WITH BROKER bos
(
   "bos_endpoint" = "http://bj.bcebos.com",
   "bos_accesskey" = "xxxxxxxxxxxxxxxxxxxxxxxxx",
   "bos_secret_accesskey"="yyyyyyyyyyyyyyyyyyyyyyyy"
);
```

Keywords

```
EXPORT
```

Best Practices

Split of subtasks

An Export job will be split into several subtasks (execution plans) for execution. How many query plans need to be executed depends on the total number of tablets and the maximum number of tablets that can be allocated to a query plan.

Because multiple query plans are executed in series, if one query plan handles more slices, the execution time of the job can be reduced.

However, if the query plan goes wrong (for example, RPC calling Broker fails, remote storage jitters, etc.), too many Tablet will lead to higher retry cost of a query plan.

Therefore, it is necessary to reasonably arrange the number of query plans and the number of slices that each query plan needs to scan, so as to balance the execution time and the success rate of execution.

Generally, it is recommended that the amount of data scanned by a query should be within 3-5 GB.

Memory limit

Usually, the query plan of an Export job has only two parts: `扫描-导出`and does not involve the calculation logic that needs too much memory. Therefore, the default memory limit of 2GB is usually enough.

However, in some scenarios, such as a query plan, too many tablets need to BE scanned on the same BE, or too many data versions of tablets may lead to insufficient memory. At this time, it is necessary to set larger memory, such as 4GB and 8GB, through`exec_mem_limit` .

Notes

- Exporting large amounts of data at once is not recommended. The maximum recommended amount of Exported data for an export job is tens of GB. Excessive export will lead to more junk files and higher retry cost. If the amount of table data is too large, it is recommended to export by partition.
- If the Export job fails to run, the `__doris_export_tmp_xxx` temporary directory generated in the remote storage and the generated files will not be deleted, and need to be deleted manually by the user.
- If the Export job runs successfully, the `__doris_export_tmp_xxx` directory generated in the remote storage may be kept or cleared according to the file system semantics of the remote storage. For example, in Baidu Object Store (BOS), after the last file in a directory is removed by rename operation, the directory will also be deleted. If the directory is not cleared, the user can clear it manually.

- The Export job only exports the data of the Base table, not the materialized view.
- The Export operation scans data, which takes up IO resources and may affect the query delay of the system.
- The maximum number of Export jobs running simultaneously in a cluster is 5. Only jobs submitted after that will be queued.

STREAM-LOAD

STREAM LOAD

Description

Stream Load transmits and loads data to Palo through HTTP protocol. This mode is mainly for uploading and loading local data of users. But it is essentially an load framework, and its HTTP interface can not only support the transmission of local data, but also support the transmission of data from memory data and pipeline data to HTTP ports.

- The public cloud user must use the HTTP protocol port of Compute Node(BE), which is 8040 by default.
- Privatized deployment users can use the HTTP protocol port of Leader Node(FE), which is 8030 by default. It must ensure that the machine network where the client is located can be connected to the machine where Compute Node is located.

This document mainly introduces the use mode of Stream Load by cURL command.

```
 curl -XPUT --location-trusted -u user:passwd \
[-H "header1: xxx" -H "header2: xxx" ...] \
-T data.txt \
http://host:port/api/{db}/{table}/_stream_load
```

- The request mode of HTTP is `PUT`
- At present, HTTP chunked and non-chunked upload methods are supported. For non-chunked mode, the Header must contain `Content-Length` to identify the length of uploaded content to ensure the integrity of data.
- Header should contain `Expect Header: 100-continue` , which can avoid unnecessary data transmission in some error scenarios.
- There are two types of target `host:port` for command:
  1. HTTP protocol port pointing to FE.In this way, the FE will directly forward the request 307 to a random BE node. The final request and data communicate directly with this BE node. This method requires that the network of client and BE node can communicate normally.
  2. HTTP protocol port pointing to BE. The request directly interacts with the BE node.

> Note: Baidu cloud Palo users can directly connect to the HTTP protocol port of Compute Node.

- Specify the database and table to be loaded in the two pathparameters `{db}`and `{table}` of the URL.
- Other parameters of the load task are specified in the Header:
  - `label`

    Specify a Label for the load task to uniquely identify the job. If not specified, the system will automatically generate a UUID as a Label.

    `-H "label: my_label1"`
  - `column_separator`

    Used to specify the column separator in the load file, which is `\t` by default. If it is an invisible character, you need to

prefix it with \x and use hexadecimal to represent the separator. For example, the delimiter `\x01` of the hive file needs to be specified as `\x01`.

`-H "column_separator: ,"`

- `columns`

  Used to specify the mapping relationship between file columns and columns in tables, and various column transformations. For a detailed introduction of this part, please refer to Mapping, transformation and filtering of columns document.

  `-H "columns: k1, k2, tmpk1, k3 = tmpk1 + 1"`

- `where`

  Filter the loaded data according to the conditions. For a detailed introduction of this part, please refer to Mapping, transformation and filtering of columns document.

  `-H "where: k1 > 100"`

- `max_filter_ratio`

  Maximum tolerance of filterable (data nonstandard, etc.) data ratio. Zero tolerance by default. The value range is 0 to 1.

  `-H "max_filter_ratio: 0.01"`

- `partitions`

  Specify the partition that needs to load data.

  `-H "partitions: p1, p2"`

- `timeout`

  Specify the timeout for load, in seconds, which is 600s by default. The setting range is 1s to 14400s.

  `-H "timeout: 120"`

- `strict_mode`

  Whether there are strict restrictions on data. The default is "False".

  `-H "strict_mode: true"`

- `timezone`

  Specify the time zone used for this load. The default is East Eight Districts. This parameter will affect the results of all functions related to time zone involved in load.

  `-H "timezone: Asia/Shanghai"`

- `exec_mem_limit`

  Load memory limits. Default to 2, in bytes.

  `-H "exec_mem_limit: 4294967296"`

- `format`

  Specify the load data format. Support `csv` and `json` , `csv` by default.

  `-H "format: json"`

- `jsonpaths`

  When the loaded data format is json, you can specify the fields in the extracted Json data through jsonpaths.

  `-H "jsonpaths: [\"$.k2\", \"$.k1\"]"`

- `strip_outer_array`

  When the loaded data format is json, `strip_outer_array` is `true` , which means that Json data is presented in the form

of array, and each element in the data will be regarded as a row of data. The default is "False".

`-H "strip_outer_array: true"`

- `json_root`

  When the format of loaded data is json, user may specify the root node of Json data by `json_root` . Palo will extract the elements of the root node through `json_root` for parsing. It is empty by default.

  `-H "json_root: $.RECORDS"`

- `merge_type`

  The default value is APPEND, which means that this load is an ordinary append write operation. MERGE and DELETE types are only applicable to Unique Key model tables. The MERGE type needs to be used with the `delete` parameter to mark the Delete Flag column. The DELETE type means that all the data loaded this time are deleted data.

  `-H "merge_type: MERGE"`

- delete: Meaningful only under MERGE type, used to specify the Delete Flag column and the conditions for marking the delete flag.

  `-H "delete: col3 = 1"`

- `function_column.sequence_col`

  Only tables for the Unique Key model. It is used to specify the column representing Sequence Col in the loaded data. It is mainly used to ensure data order during loading.

  `-H "function_column.sequence_col: col3"`

- `fuzzy_parse`

  When the loaded data format is Json array, and the field order of each row in the array is completely consistent. This parameter can be turned on to speed up the load. Generally it is used in combination with `strip_outer_array: true` . Details can be found in JSON FormatDataLoad Instructions.

  `-H "fuzzy_parse: true"`

Example

1. Load the local file testData and specify the timeout period

```
 curl --location-trusted -u admin -H "label:label1" -H "timeout:100" -T testData
http://host:port/api/example_db/my_table/_stream_load
```

2. Load the local file testData and filter the data according to the conditions

```
 curl --location-trusted -u admin -H "label:label2" -H "where: k1=20180601" -T testData
http://host:port/api/example_db/my_table/_stream_load
```

3. Load the local file testData and set the maximum allowable error rate

```
 curl --location-trusted -u admin -H "label:label3" -H "max_filter_ratio:0.2" -T testData
http://host:port/api/example_db/my_table/_stream_load
```

4. Load the local file testData and specify the the column mapping relationship

```
 curl --location-trusted -u admin -H "label:label4" -H "max_filter_ratio:0.2" -H "columns: k2, k1, v1" -T testData
http://host:port/api/example_db/my_table/_stream_load
```

5. Load the local file testData and specify the partition and maximum allowable error rate

```
curl --location-trusted -u admin -H "label:label5" -H "max_filter_ratio:0.2" -H "partitions: p1, p2" -T testData
http://host:port/api/example_db/my_table/_stream_load
```

6. Load using streaming method

```
seq 1 10 | awk '{OFS="\t"}{print $1, $1 * 10}' | curl --location-trusted -u admin -T -
http://host:port/api/example_db/my_table/_stream_load
```

7. Load a table with HLL columns

```
curl --location-trusted -u admin -H "columns: k1, k2, v1=hll_hash(k1), v2=hll_empty()" -T testData
http://host:port/api/example_db/my_table/_stream_load
```

8. Load a table with BITMAP columns

```
curl --location-trusted -u admin -H "columns: k1, k2, v1=to_bitmap(k1), v2=bitmap_empty()" -T testData
http://host:port/api/example_db/my_table/_stream_load
```

9. Load Json data, using simple mode. That is, the field name in Json data is the column name.

   The table structure is:

```
category    varchar(512)
author      varchar(512)
title       varchar(512)
price       double
```

   Json data:

```
{"category":"C++","author":"avc","title":"C++ primer","price":895}
```

   Load command:

```
curl --location-trusted -u admin -H "label:label10" -H "format: json" -T testData
http://host:port/api/example_db/my_table/_stream_load
```

10. Load Json data and extract fields using jsonpath

json data format:

```
{"category":"xuxb111","author":"1avc","title":"SayingsoftheCentury","price":895},
```

Use jsonpath to extract three fields: `category, author, price` .

```
curl --location-trusted -u admin -H "columns: category, price, author" -H "label:123" -H "format: json" -H "jsonpaths: [\"$.category\",\"$.price\",\"$.author\"]" -H "strip_outer_array: true" -T testData http://host:port/api/example_db/my_table/_stream_load
```

11. Load Json data, specify Json document root node, and flatten array.

    json data format:

```
{
"RECORDS":[
    {"category":"11","title":"SayingsoftheCentury","price":895,"timestamp":1589191587},
    {"category":"22","author":"2avc","price":895,"timestamp":1589191487},
    {"category":"33","author":"3avc","title":"SayingsoftheCentury","timestamp":1589191387}
    ]
}
```

    Use jsonpath to extract three fields: `category, author, price` .

```
curl --location-trusted -u admin -H "columns: category, price, author" -H "label:label12" -H "format: json" -H "jsonpaths: [\"$.category\",\"$.price\",\"$.author\"]" -H "strip_outer_array: true" -H "json_root: $.RECORDS" -T testData http://host:port/api/example_db/my_table/_stream_load
```

12. Use DELETE mode to delete the same data as this batch of loaded keys

```
curl --location-trusted -u admin -H "merge_type: DELETE" -T testData http://host:port/api/example_db/my_table/_stream_load
```

13. Use MERGE mode. Delete the columns in a batch of data that match the data whose `flag` column is true, and append other rows normally

```
curl --location-trusted -u admin -H "column_separator:," -H "columns: siteid, citycode, username, pv, flag" -H "merge_type: MERGE" -H "delete: flag=1" -T testData http://host:port/api/example_db/my_table/_stream_load
```

14. Load data into a Unique Key model table with a Sequence Col column

```
curl --location-trusted -u admin -H "columns: k1,k2,source_sequence,v1,v2" -H "function_column.sequence_col: source_sequence" -T testData http://host:port/api/example_db/my_table/_stream_load
```

Keywords

```
STREAM, LOAD
```

Best Practices

1. View load task status

   Stream Load is a synchronous load process, and the successful execution of statements means the successful load of data. The execution result of the load will be returned synchronously through HTTP return value. And present it in Json format.


```
{
  "TxnId": 17,
  "Label": "707717c0-271a-44c5-be0b-4e71bfeacaa5",
  "Status": "Success",
  "Message": "OK",
  "NumberTotalRows": 5,
  "NumberLoadedRows": 5,
  "NumberFilteredRows": 0,
  "NumberUnselectedRows": 0,
  "LoadBytes": 28,
  "LoadTimeMs": 27,
  "BeginTxnTimeMs": 0,
  "StreamLoadPutTimeMs": 2,
  "ReadDataTimeMs": 0,
  "WriteDataTimeMs": 3,
  "CommitAndPublishTimeMs": 18
}
```


   The field definitions are as follows:

   - TxnId：Load transaction ID, which is automatically generated by the system and unique globally.
   - Label：Load Label, if not specified, the system will generate a UUID.
   - Status：

     Load result. There are the following values:

     - Success：Indicates that the load is successful and the data is visible.
     - Publish Timeout：This status also indicates that the load has been completed, but the data may be visible later.
     - Label Already Exists：The Label is duplicate, and the Label needs to be replaced.
     - Fail：Load failed.

- ExistingJobStatus :

  Status of load job corresponding to existing Label.

  This field will only be displayed when the Status is "Label Already Exists". The user can know the status of the existing load job corresponding to Label through this status. "RUNNING" means that the job is still executing, and "FINISHED" means that the job is successful.

- Message : Load error messages.
- NumberTotalRows : Load the total number of rows processed.
- NumberLoadedRows : Number of rows successfully loaded.
- NumberFilteredRows : Number of rows with unqualified data quality.
- NumberUnselectedRows : The number of rows filtered by the where condition.
- LoadBytes : Number of bytes loaded.
- LoadTimeMs : Load finish time, in milliseconds
- BeginTxnTimeMs : Time spent requesting FE to start a transaction, in milliseconds.
- StreamLoadPutTimeMs : It takes time to ask FE for the load data execution plan, in milliseconds.
- ReadDataTimeMs : Time spent on reading data, in milliseconds.
- WriteDataTimeMs : Time spent onexecuting the write data operation, in milliseconds.
- CommitAndPublishTimeMs : Time taken to submit and publish a transaction to the Fe request, in milliseconds.
- ErrorURL : If there is data quality problem, visit this URL to view the specific error line.

2. How to correctly submit Stream Load job and process the returned result.

   Stream Load is a synchronous load operation, so the user needs to wait for the return result of the command synchronously and decide the next processing mode according to the return result.

   The user's primary concern is to return the `Status` field in the result.

   If it is `Success` , everything is normal and other operations can be performed later.

   If there are a large number of `Publish Timeout`, in the returned results, it may indicate that some resources (such as IO) in the cluster are short at present, so that the loaded data cannot take effect finally. Load task in `Publish Timeout` status has been successful, so there is no need to retry. however, it is suggested to slow down or stop the submission of new load task and observe the cluster load.

   If the returned result is `Fail` , and the problem should be checked according to specific reasons. Once resolved, you can try again using the same Label.

   In some cases, the user's HTTP connection may be abnormally disconnected, resulting in that the final return result cannot be obtained. At this time, you can resubmit the load task with the same Label, and the resubmitted task may have the following results:

   1. `Status` is `Success` , `Fail` or `Publish Timeout`. At this time, it can be processed according to the normal process.
   2. `Status` is `Label Already Exists`. continue to view the `ExistingJobStatus` field. If the value of this field is`FINISHED`, it means that the load task corresponding to this Label has been successful, so there is no need to try again. If it is `RUNNING` , it means that the load task corresponding to this Label is still running. At this time, the same Label should be used at regular intervals (for example, 10 seconds) to continue submitting repeatedly until the `Status` is no longer the `Label Already Exists` , or the `ExistingJobStatus` field value is `FINISHED` .

3. Cancel the load task

Load tasks that have been submitted but not finished can be cancelled by the CANCEL LOAD command. After cancellation, the written data will also be rolled back and will not take effect.

4. Label, load transaction, multi-table atomicity

   All load tasks in Palo are atomic. Moreover, the atomicity can be guaranteed by loading multiple tables in the same load task. At the same time, Palo can also ensure that data load is not lost or duplicated through the mechanism of Label. Specific instructions can be found in Load Transaction and Atomicity document.

5. Column mapping, derived columns and filtering

   Palo can support very rich column conversion and filtering operations in load statements. 支Most built-in functions and UDFs are supported. For how to use this function correctly, please refer to Column mapping, derived columns and filtering document.

6. Error data filtering

   Palo's load task can tolerate some malformed data. Tolerance ratecan be set by `max_filter_ratio` , which takes 0 by default, indicating that, when there is an error data, the whole load task will fail.If the user wants to ignore some data rows with problems, the secondary parameter can be set to a value between 0 and 1, and Palo will automatically skip the rows with incorrect data format.

   For some calculation methods of tolerance rate, please refer to Mapping, transformation and filtering of columns document.

7. strict mode

   `strict_mode` property is used to set whether the load task runs in strict mode. This format will affect the results of column mapping, transformation and filtering. For a detailed description of the strict mode, please refer to Strict Mode document.

8. Timeout period

   The default timeout for Stream Load is 10 minutes, starting from the time the task is submitted. If it is not completed within the timeout period, the task will fail.

9. Data volume and task number limit

   Stream Load is suitable for loading data within a few GB, because the data is processed by single thread transmission, so the performance of loading too large data cannot be guaranteed. When a large amount of local data needs to be loaded, multiple load tasks can be submitted in parallel.

   Palo also limits the number of load tasks running simultaneously in the cluster, usually ranging from 10 to 20. Load jobs submitted later will be rejected.

RESTORE

RESTORE Description

This statement is used to restore the back backed up by BACKUP command to specified database.

This command is an asynchronous operation. After successful submission, the progress should be viewed by SHOW RESTORE command.

Only OLAP-type tables can be restored.

```
RESTORE SNAPSHOT [db_name.]snapshot_name
FROM `repository_name`
ON (
    `table_name` [PARTITION (`p1`, ...)] [AS `tbl_alias`],
    ...
)
PROPERTIES ("key"="value", ...);
```

- `snapshot_name` : the Snapshot name that can be viewed by `SHOW SNAPSHOT` command.
- `repository_name` : repository name.
- `ON`

  `ON` clause specifies the tables and partitions in the snapshot that need to be backed up. In which the table name and partition name must be the names in the backup snapshot, and `tbl_alias` can specify aliases for the table. The final recovered table will adopt this alias. Partition name cannot be modified. If no partition is specified, all partitions of the table are restored by default. The specified table and partition must already exist in the respository backup.
- `PROPERTIES` : Specify properties related to recovery operation
  - `backup_timestamp` : Specify which time version of the corresponding backup snapshot to restore; required. This information can be obtained by `SHOW SNAPSHOT` command.
  - `replication_num` : Specify the number of replication of the recovered table or partition, which is 3 by default. Specify the number of replication of the recovered table or partition. At the same time, the number of Compute Node nodes must be greater than or equal to the specified number of replication.
  - `timeout` : Task timeout period, default to one day, in seconds.

Example

1. Back up the table backup_tbl in snapshot_1 from example_repo to the database example_db1 with the time version of "2020-05-04-16-45-08". Restore to 1 replication:

```
 RESTORE SNAPSHOT example_db1.`snapshot_1`
FROM `example_repo`
ON ( `backup_tbl` )
PROPERTIES
(
    "backup_timestamp"="2020-05-04-16-45-08",
    "replication_num" = "1"
);
```

2. Partition P1 and p2 of backup_tbl in backup snapshot_2 and backup_tbl2 are restored from example_repo to database example_db1, and renamed as new_tbl with the time version of "2020-05-04-17-11-01". The default recovery is 3 replications:

```
 RESTORE SNAPSHOT example_db1.`snapshot_2`
FROM `example_repo`
ON
(
    `backup_tbl` PARTITION (`p1`, `p2`),
    `backup_tbl2` AS `new_tbl`
)
PROPERTIES
(
    "backup_timestamp"="202-05-04-17-11-01"
);
```

Keywords

```
RESTORE, SNAPSHOT
```

Best Practices

1. There can only be one ongoing recovery operation under the same database.
2. The table backed up in the respository can be restored to replace the existing table with the same name in the database, but the table structure of the two tables must be completely consistent. Table structure includes: Table names, columns, partitions, materialized views, and so on.
3. When a partial partition of the recovery table is specified, the system checks whether the partition range can match.
4. Efficiency of recovery operation:

   When the cluster size is the same, the time consumption of recovery operation is basically equal to that of backup operation. To speed up the recovery operation, user can restore only one replication by setting `replication_num` and then adjust the number of replications through ALTER TABLE PROPERTY to make up the number of replications.

ROUTINE-LOAD

ROUTINE LOAD Description

Routine Load function, which supports users to submit a resident load task and load data into Palo by continuously reading data from the specified data source.

At present, it only supports loading CSV or Json format data from Kakfa by means of no authentication or SSL authentication.

Syntax:

```
CREATE ROUTINE LOAD [db.]job_name ON tbl_name
[merge_type]
[load_properties]
[job_properties]
FROM data_source [data_source_properties]
```

- `[db.]job_name`

  The name of the load job. in the same database, only one job with the same name can run.

- `tbl_name`

  Specify the name of the table to be loaded.

- `merge_type`

  Data merger type. The default value is APPEND, which means that the data loaded are ordinary append write operations. MERGE and DELETE types are only applicable to Unique Key model tables. The MERGE type needs to be used with the [DELETE ON] statement to mark the Delete Flag column. The DELETE type means that all the data loaded this time are deleted data.

- load_properties

  Used to describe loaded data, which consists of:

  ```
  [column_separator],
  [columns_mapping],
  [preceding_filter],
  [where_predicates],
  [partitions],
  [DELETE ON],
  [ORDER BY]
  ```

  - `column_separator`

    Specify column separator, default is `\t`

    `COLUMNS TERMINATED BY ","`

  - `columns_mapping`

    Used to specify the mapping relationship between file columns and columns in tables, and various column transformations. For a detailed introduction of this part, please refer to the [Mapping, Transformation and Filtering of Columns] document.

    `(k1, k2, tmpk1, k3 = tmpk1 + 1)`

- `preceding_filter`

  Filter raw data. For a detailed introduction of this part, please refer to the [Mapping, Transformation and Filtering of Columns] document.

- `where_predicates`

  Filter the loaded data according to the conditions. For a detailed introduction of this part, please refer to the [Mapping, Transformation and Filtering of Columns] document.

  `WHERE k1 > 100 and k2 = 1000`

- `partitions`

  Specify which partition of the destination table to load. If not specified, it will be automatically loaded into the corresponding partition.

  `PARTITION(p1, p2, p3)`

- `DELETE ON`

  It needs to be used together with MEREGE load mode, only for tables of Unique Key model. And it is used to specify the column and calculation relation of Delete Flag in loaded data.

  `DELETE ON v3 >100`

- `ORDER BY`

  Only tables for the Unique Key model. It is used to specify the column representing Sequence Col in the loaded data. It is mainly used to ensure data order during loading.

- `job_properties`

  Used to specify general parameters for routine load jobs.

```
PROPERTIES (
  "key1" = "val1",
  "key2" = "val2"
)
```

  At present we support the following parameters:

  1. `desired_concurrent_number`

     Expected degree of concurrency. A routine load job will be divided into several subtasks. This parameter specifies the maximum number of tasks that a job can perform at the same time. It must be greater than 0 and is 3 by default.

     This concurrency is not the actual concurrency. The actual concurrency will be comprehensively considered through the number of nodes in the cluster, the load situation and the data source situation.

     `"desired_concurrent_number" = "3"`

  2. `max_batch_interval/max_batch_rows/max_batch_size`

     These three parameters respectively represent:

     1. Maximum execution time of each subtask, in seconds. It ranges from 5 to 60 and is 10 by default.
     2. The maximum number of rows read per subtask, which must be greater than or equal to 200000 and is 200000 by default.
     3. The maximum number of lines read per subtask. The unit is bytes and the range is 100MB to 1GB. it takes 100MB by default.

     These three parameters are used to control the execution time and throughput of a subtask. When any one reaches the threshold, the task ends.

```
 "max_batch_interval" = "20",
"max_batch_rows" = "300000",
"max_batch_size" = "209715200"
```

3. `max_error_number`

   The maximum number of error rows allowed in the sampling window, which must be greater than or equal to 0. The default value is 0, that is, error lines are not allowed.

   Sampling window is `max_batch_rows * 10`。That is, if the number of error rows is larger than `max_error_number` in the sampling window, routine jobs will be suspended, and manual intervention is needed to check data quality problems.

   Rows filtered out by the where condition are not considered as error rows.

4. `strict_mode`

   Whether strict mode is turned on or not is turned off by default. If it is turned on, the column type transformation of non-empty original data will be filtered if the result is NULL. Specify the method as follows:

   `"strict_mode" = "true"`

5. `timezone`

   Specify the time zone used by the load job. The default is to use the timezone parameter of Session. This parameter will affect the results of all functions related to time zone involved in load.

6. `format`

   Specify the load data format, which is csv by default and supports json format.

7. `jsonpaths`

   When the loaded data format is json, you can specify the fields in the extracted Json data through jsonpaths.

   `-H "jsonpaths: [\"$.k2\", \"$.k1\"]"`

8. `strip_outer_array`

   When the loaded data format is json, strip_outer_array is true, which means that Json data is presented in the form of array, and each element in the data will be regarded as a row of data. The default is "False".

   `-H "strip_outer_array: true"`

9. `json_root`

   When the loaded data format is json, you can specify the fields in the root Json data through json_root. Palo will extract the elements of the root node through json_root for parsing. It is empty by default.

   `-H "json_root: $.RECORDS"`

- `FROM data_source [data_source_properties]`

  Data source type. Currently support:

```
 FROM KAFKA
(
   "key1" = "val1",
   "key2" = "val2"
)
```

  `data_source_properties` supports the following data source properties:

  1. `kafka_broker_list`

     Broker connection information for Kafka, in the format ip:host. Multiple broker are separated by commas.

     `"kafka_broker_list" = "broker1:9092,broker2:9092"`

2. `kafka_topic`

   Specify the topic of Kafka to subscribe to.

   `"kafka_topic" = "my_topic"`

3. `kafka_partitions/kafka_offsets`

   Specify the kafka partition that need to be subscribed, and the corresponding starting offset of each partition.

   Offset can specify a specific offset from 0 or more, or:

   - `OFFSET_BEGINNING`: Subscribe from where there is data.
   - `OFFSET_END`: Subscribe from the end.

   If not specified, all partition under topic will be subscribed by default from `OFFSET_END` .

   ```
   "kafka_partitions" = "0,1,2,3",
   "kafka_offsets" = "101,0,OFFSET_BEGINNING,OFFSET_END"
   ```

4. `property`

   Specify custom kafka parameters. The function is equivalent to the "--property" parameter in kafka shell.

   When the value of the parameter is a file, you need to add keyword "FILE:" before the value.

   For information on how to create a file, please refer to CREATE FILE command document.

   See the CONFIGURATION items on the client side in the official configuration document of librdkafka for more supported custom parameters. Such as:

   ```
   "property.client.id" = "12345",
   "property.ssl.ca.location" = "FILE:ca.pem"
   ```

   1. The following parameters need to be specified when connecting Kafka using SSL:

      ```
      "property.security.protocol" = "ssl",
      "property.ssl.ca.location" = "FILE:ca.pem",
      "property.ssl.certificate.location" = "FILE:client.pem",
      "property.ssl.key.location" = "FILE:client.key",
      "property.ssl.key.password" = "abcdefg"
      ```

      In which:

      `property.security.protocol` and `property.ssl.ca.location` are required to indicate that the connection method is SSL and the location of the CA certificate.

      If client authentication is enabled on Kafka server side, it is also necessary to set:

      ```
      "property.ssl.certificate.location"
      "property.ssl.key.location"
      "property.ssl.key.password"
      ```

      Used to specify public key, private key and password of private key of client respectively.

   2. Specify the default starting offset of kafka partition

      If `kafka_partitions/kafka_offsets` is not specified, all partitions are consumed by default.

      At this time, the start offset of `kafka_default_offsets` can be specified, which is `OFFSET_END` by default, that is, subscribe from the end.

      Example:

      ```
      "property.kafka_default_offsets" = "OFFSET_BEGINNING"
      ```

Example

1. Create a Kafka routine load task named test1 for example_tbl of example_db. Specify column separator, group.id and client.id, automatically consume all partitions by default, and subscribe from the position where there is data (OFFSET_BEGINNING)

```
CREATE ROUTINE LOAD example_db.test1 ON example_tbl
COLUMNS TERMINATED BY ",",
COLUMNS(k1, k2, k3, v1, v2, v3 = k1 * 100)
PROPERTIES
(
    "desired_concurrent_number"="3",
    "max_batch_interval" = "20",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200",
    "strict_mode" = "false"
)
FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic",
    "property.group.id" = "xxx",
    "property.client.id" = "xxx",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

2. Create a Kafka routine load task named test1 for example_tbl of example_db. Load task is in strict mode.

```
 CREATE ROUTINE LOAD example_db.test1 ON example_tbl
COLUMNS(k1, k2, k3, v1, v2, v3 = k1 * 100),
PRECEDING FILTER k1 = 1,
WHERE k1 > 100 and k2 like "%palo%"
PROPERTIES
(
    "desired_concurrent_number"="3",
    "max_batch_interval" = "20",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200",
    "strict_mode" = "false"
)
FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic",
    "kafka_partitions" = "0,1,2,3",
    "kafka_offsets" = "101,0,0,200"
);
```

3. Load data from Kafka cluster through SSL authentication. Set the client.id parameter at the same time. Load task is in non-strict mode and time zone is Africa/Abidjan

```
 CREATE ROUTINE LOAD example_db.test1 ON example_tbl
COLUMNS(k1, k2, k3, v1, v2, v3 = k1 * 100),
WHERE k1 > 100 and k2 like "%palo%"
PROPERTIES
(
    "desired_concurrent_number"="3",
    "max_batch_interval" = "20",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200",
    "strict_mode" = "false",
    "timezone" = "Africa/Abidjan"
)
FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic",
    "property.security.protocol" = "ssl",
    "property.ssl.ca.location" = "FILE:ca.pem",
    "property.ssl.certificate.location" = "FILE:client.pem",
    "property.ssl.key.location" = "FILE:client.key",
    "property.ssl.key.password" = "abcdefg",
    "property.client.id" = "my_client_id"
);
```

4. Load Json format data. By default, field names in Json are used as column name mappings. Specify to load into three partitions, 0, 1 and 2, with the initial offset of 0

```
CREATE ROUTINE LOAD example_db.test_json_label_1 ON table1
COLUMNS(category,price,author)
PROPERTIES
(
    "desired_concurrent_number"="3",
    "max_batch_interval" = "20",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200",
    "strict_mode" = "false",
    "format" = "json"
)
FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic",
    "kafka_partitions" = "0,1,2",
    "kafka_offsets" = "0,0,0"
);
```

5. Load Json data, extract fields through Jsonpaths, and specify the root node of Json document

```
 CREATE ROUTINE LOAD example_db.test1 ON example_tbl
COLUMNS(category, author, price, timestamp, dt=from_unixtime(timestamp, '%Y%m%d'))
PROPERTIES
(
    "desired_concurrent_number"="3",
    "max_batch_interval" = "20",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200",
    "strict_mode" = "false",
    "format" = "json",
    "jsonpaths" = "[\"$.category\",\"$.author\",\"$.price\",\"$.timestamp\"]",
    "json_root" = "$.RECORDS"
    "strip_outer_array" = "true"
)
FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic",
    "kafka_partitions" = "0,1,2",
    "kafka_offsets" = "0,0,0"
);
```

6. Create a Kafka routine load task named test1 for example_tbl of example_db. And use conditional filtering.

```
 CREATE ROUTINE LOAD example_db.test1 ON example_tbl
WITH MERGE
COLUMNS(k1, k2, k3, v1, v2, v3),
WHERE k1 > 100 and k2 like "%palo%",
DELETE ON v3 >100
PROPERTIES
(
    "desired_concurrent_number"="3",
    "max_batch_interval" = "20",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200",
    "strict_mode" = "false"
)
FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic",
    "kafka_partitions" = "0,1,2,3",
    "kafka_offsets" = "101,0,0,200"
);
```

7. Load data into a Unique Key model table with a sequence column

```
 CREATE ROUTINE LOAD example_db.test_job ON example_tbl
COLUMNS TERMINATED BY ",",
COLUMNS(k1,k2,source_sequence,v1,v2),
ORDER BY source_sequence
PROPERTIES
(
    "desired_concurrent_number"="3",
    "max_batch_interval" = "30",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200"
) FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic",
    "kafka_partitions" = "0,1,2,3",
    "kafka_offsets" = "101,0,0,200"
);
```

Keywords

```
CREATE, ROUTINE, LOAD
```

Best Practices

TBD

SELECT-INTO-OUTFILE

SELECT INTO OUTFILE

Description

This command is used to export the result set of SQL output to local disk or to remote storage through Broker.

```
 query_stmt
INTO OUTFILE "file_path"
[format_as]
[properties]
```

- `query_stmt`

Any query statement.

- file_path

  Path to file storage and file prefix. For example:

  ```
   bos://my_bucket/my_file_          # remote catalog
  file:///to/loca/path/my_file_      # local catalog
  ```

  The final file name will consist of my_file_ + fileS/N + file format suffix .

  The file serial number starts from 0, and the number is the number of files divided. For example:

  ```
   my_file_0.csv
  my_file_1.csv
  my_file_2.csv
  ```

  > If the local file mode is used, the exported file will be stored on a random Compute Node inthe cluster. Specific node information is known in the URL of the returned result.

- format_as

  Export file format. Only support CN currently

  ```
   FORMAT AS CSV
  ```

- properties

  Related properties. The properties required by broker must start with broker. . For example:

  ```
   (
     "broker.prop_key" = "prop_val",
  )
  ```

  其他参数如：

  - column_separator：Column separator, only applicable to CSV format. Default to \t.
  - line_delimiter：line delimiter, only applicable to CSV format. Default to \n.
  - max_file_size：The maximum size of a single file. Default to 1GB. The value range is between 5MB and 2GB. Files larger than this size will be split.
  - success_file_name：Whether to generate an empty file identifier after success. File name is "my_file_file_name". In which, mysq_file is the prefix specified in file_path , file_name is the value of the parameter.

Description of return results:

If it exports and returns normally, the result is as follows:

```
mysql> select * from tbl1 limit 10 into outfile "file:///home/work/path/result_";
+------------+-----------+----------+-------------+
| FileNumber | TotalRows | FileSize | URL         |
+------------+-----------+----------+-------------+
|          1 |         2 |        8 | 192.168.1.10 |
+------------+-----------+----------+-------------+
1 row in set (0.05 sec)
```

- `FileNumber` : The number of files finally generated.
- `TotalRows` : Number of rows in result set.
- `FileSize` : Total size of export file, in bytes.
- `URL` : It exporting to local disk, it shows which Compute Node to export to.

If there is error in execution, an error message will be returned, such as:

```
mysql> SELECT * FROM tbl INTO OUTFILE ...
ERROR 1064 (HY000): errCode = 2, detailMessage = Open broker writer failed ...
```

Example

1. Export simple query results to file `bos://my_bucket/result_`. Specify that the export format is CSV. Use `my_broker` and set kerberos authentication information. The specified column separator is `,` , and the line delimiter is `\n`.

```
SELECT * FROM tbl
INTO OUTFILE "bos://my_bucket/result_"
FORMAT AS CSV
PROPERTIES
(
    "broker.name" = "bos",
    "broker.bos_endpoint" = "http://bj.bcebos.com",
    "broker.bos_accesskey" = "xxxxxxxxxxxxxxxxxxxxxxxxx",
    "broker.bos_secret_accesskey" = "yyyyyyyyyyyyyyyyyyyyyyyy",
    "column_separator" = ",",
    "line_delimiter" = "\n",
    "max_file_size" = "100MB"
);
```

   If the final generated file is not larger than 100MB, then : `result_0.csv`.

   If it is larger than 100MB, it may be `result_0.csv, result_1.csv, ...`.

2. Export the query result of CTE statement to file `bos://my_bucket/result_`.

```
WITH
x1 AS
(SELECT k1, k2 FROM tbl1),
x2 AS
(SELECT k3 FROM tbl2)
SELEC k1 FROM x1 UNION SELECT k3 FROM x2
INTO OUTFILE "bos://my_bucket/result_"
PROPERTIES
(
    "broker.name" = "bos",
    "broker.bos_endpoint" = "http://bj.bcebos.com",
    "broker.bos_accesskey" = "xxxxxxxxxxxxxxxxxxxxxxxxx",
    "broker.bos_secret_accesskey" = "yyyyyyyyyyyyyyyyyyyyyyyy"
);
```

If the final generated file is not larger than 1GB, then : `result_0.csv`.

If it is larger than 1GB, it may be `result_0.csv, result_1.csv, ...`.

3. Export the query result of UNION statement to file bos: `bos://my_bucket/result.txt`. And an empty file identifier is generated after successful exporting.

```
SELECT k1 FROM tbl1 UNION SELECT k2 FROM tbl1
INTO OUTFILE "bos://bucket/result_"
PROPERTIES
(
    "broker.name" = "my_broker",
    "broker.bos_endpoint" = "http://bj.bcebos.com",
    "broker.bos_accesskey" = "xxxxxxxxxxxxxxxxxxxxxxxxx",
    "broker.bos_secret_accesskey" = "yyyyyyyyyyyyyyyyyyyyyyyy",
    "success_file_name" = "SUCCESS"
);
```

If the final generated file is not larger than 1GB, then : `result_0.parquet`.

If it is larger than 1GB, it may be `result_0.parquet, result_1.parquet, ...`.

A successful file is identified as `result_SUCCESS`.

4. Export results to local disk.

```
SELECT k1 FROM tbl1 UNION SELECT k2 FROM tbl1
INTO OUTFILE "file:///local/path/result_"
PROPERTIES
(
    "column_separator" = ",",
    "line_delimiter" = "\n",
    "max_file_size" = "100MB"
);
```

Keywords

SELECT, INTO, OUTFILE

Best Practices

1. Export data quantity and export efficiency

   The function is essentially to execute a SQL query command. The final result is single-threaded output. Therefore, the time consumption of the whole export includes the time consumption of the query itself and the time consumption of writing the final result set. If the query is large, it is necessary to set the session variable query_timeout to extend the query timeout appropriately.

2. Management of export files

   Palo does not manage exported files, Including those files that were successfully exported or remained after the export failed, all need to be handled by users themselves.

3. Export to local files

   The function of exporting to local files is not applicable to public cloud users, but only to users deployed in privatization. And the default user has complete control authority over the cluster nodes. Palo does not check the legality of the export path filled in by the user. If the process user of Palo does not have write access to the path, or the path does not exist, an error will be reported. At the same time, for security reasons, if a file with the same name already exists in this path, the export will also fail.

   Palo does not manage files exported locally, and does not check disk space, etc. These files need to be managed by users themselves, such as cleaning.

4. Guarantee of result integrity

   This command is a synchronous command, so it is possible that the task connection is disconnected during the execution, and it is impossible to live. Whether the exported data ends normally or is complete. At this time, user can use the `success_file_name` parameter to request that a successful file identifier be generated in the directory after the task is successful. And user can judge whether the export ends normally through this file.

INSERT

INSERT

Description

```
INSERT INTO table_name
[ PARTITION (p1, ...) ]
[ WITH LABEL label]
[ (column [, ...]) ]
{ VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }
```

- `table_name`

  The name of the table that needs to load data.

- `PARTITION`

  Specify the partition that needs to load data.

- `WITH LABEL`

  Specify a Label for this INSERT operation. If not specified, the system will automatically generate a random ID.

- `column`

  Specify the target column order

- `VALUES | query`

  INSERT operation supports two ways. One is to write single-line data through the VALUES statement, that is, constant expression. The other is to use Query statement to query data from other tables and load them.

Example

1. Load a row of data into the `test` table. The first sentence and the second sentence have the same effect. When no target column is specified, the column order in the table is used as the default target column.

   The meaning expressed in the third and fourth statements is the same, and the default value of `c2` column is used to complete the data load.

2. Load multiple rows of data into the `test`table

```
INSERT INTO test VALUES (1, 2), (3, 2 + 2);

INSERT INTO test (c1, c2) VALUES (1, 2), (3, 2 * 2);

INSERT INTO test (c1) VALUES (1), (3);

INSERT INTO test (c1, c2) VALUES (1, DEFAULT), (3, DEFAULT);
```

The first statement and the second statement have the same effect, and load two pieces of data into the`test` table at one time

The effect of the third and fourth statements is known, and the default value of `c2`column is used to load two pieces of data into the`test` table

3. Load a query statement result into the `test` table

```
INSERT INTO test SELECT * FROM test2;

INSERT INTO test (c1, c2) SELECT * from test2;

INSERT INTO tbl1 WITH LABEL label1
WITH cte1 AS (SELECT * FROM tbl1), cte2 AS (SELECT * FROM tbl2)
SELECT k1 FROM cte1 JOIN cte2 WHERE cte1.k1 = 1;
```

4. Load a query statement result into the `test` table and specify the partition and label

```
INSERT INTO test PARTITION(p1, p2) WITH LABEL `label1` SELECT * FROM test2;
INSERT INTO test WITH LABEL `label1` (c1, c2) SELECT * from test2;
```

Keywords

```
INSERT
```

Best Practices

1. View the returned results

   INSERT operation is a synchronous operation, and the result returned indicates the end of the operation. Users need to carry out corresponding processing according to different returned results.

   1. Execution succeeded, and the result set was empty

      If the result set of the select statement corresponding to insert is empty, it returns as follows:

```
mysql> insert into tbl1 select * from empty_tbl;
Query OK, 0 rows affected (0.02 sec)
```

`Query OK` indicates successful execution.`0 rows affected` indicates that no data has been loaded.

2. Execution succeeded, and the result set was not empty

   When the result set is not empty. The returned results can be divided into the following situations:

   1. Insert successfully executed and visible:

```
mysql> insert into tbl1 select * from tbl2;
Query OK, 4 rows affected (0.38 sec)
{'label':'insert_8510c568-9eda-4173-9e36-6adc7d35291c', 'status':'visible', 'txnId':'4005'}

mysql> insert into tbl1 with label my_label1 select * from tbl2;
Query OK, 4 rows affected (0.38 sec)
{'label':'my_label1', 'status':'visible', 'txnId':'4005'}

mysql> insert into tbl1 select * from tbl2;
Query OK, 2 rows affected, 2 warnings (0.31 sec)
{'label':'insert_f0747f0e-7a35-46e2-affa-13a235f4020d', 'status':'visible', 'txnId':'4005'}

mysql> insert into tbl1 select * from tbl2;
Query OK, 2 rows affected, 2 warnings (0.31 sec)
{'label':'insert_f0747f0e-7a35-46e2-affa-13a235f4020d', 'status':'committed', 'txnId':'4005'}
```

   `Query OK` indicates successful execution.`4 rows affected` indicates that total 4 rows of data have been loaded.`2 warnings` indicates the number of rows filtered.

   At the same time, a json string is returned:

```
 {'label':'my_label1', 'status':'visible', 'txnId':'4005'}
{'label':'insert_f0747f0e-7a35-46e2-affa-13a235f4020d', 'status':'committed', 'txnId':'4005'}
{'label':'my_label1', 'status':'visible', 'txnId':'4005', 'err':'some other error'}
```

`label` is a user-specified label or an automatically generated label. Label is the id of the Insert Into load job. Every load job has a unique Label within a single database.

`status` indicates whether the loaded data is visible. If visible, show `visible`; if not, show `committed`。

`txnId` is the id of the load transaction corresponding to this insert.

`err` field displays some other unexpected errors.

When it is required to view the filtered rows, the user can use the following statement

```
show load where label="xxx";
```

The URL in the returned result can be used to query the wrong data. Details can be found in the summary of **View Error Line.**

Invisible data is a temporary state, and this batch of data will eventually be visible

The visible status of this batch of data can be viewed through the following statement:

```
show transaction where id=4005;
```

If the `TransactionStatus` column in the returned result is `visible`, the representation data is visible.

3. Execution failed

   Execution failure means that no data has been successfully loaded, and returns as follows:

```
mysql> insert into tbl1 select * from tbl2 where k1 = "a";
ERROR 1064 (HY000): all partitions have no load data. url: http://10.74.167.16:8042/api/_load_error_log?file=__shard_2/error_log_insert_stmt_ba8bb9e158e4879-ae8de8507c0bf8a2_ba8bb9e158e4879_ae8de8507c0bf8a2
```

In which, `ERROR 1064 (HY000): all partitions have no load data` shows the failure reason. The following url can be used to query the wrong data:

```
show load warnings on "url";
```

Specific error lines can be viewed.

2. Timeout period

   The timeout for INSERT operation is determined by Session variable `query_timeout` , which is 5 min by default. If it times out, the job will be cancelled.

3. Label and atomicity

   INSERT operation can also guarantee the atomicity of load. Please refer to Load transactions and atomicity document.

   When `CTE(Common Table Expressions)` needs to be used as the query part in insert operation, `WITH LABEL` and `column` must be specified.

4. Filtering threshold

   Unlike other load methods, INSERT operation cannot specify （`max_filter_ratio`). The default filtering threshold is 1, that is, all lines with errors can be ignored.

   For business scenarios that require data that cannot be filtered, user can set Session variable `enable_insert_strict` as `true` to ensure that the `INSERT` will not be executed successfully when when data is filtered out.

5. Performance problem

   It is not suggested to insert a single line by using the `VALUES` method. If it must be used in this way, please combine multiple rows of data into one INSERT statement for batch submission.

Information View Statement

SHOW-FILE

SHOW FILE

Description

This statement is used to show the file created by the CREATE FILE command in a database.

```
SHOW FILE [FROM database];
```

Description of return results:

- FileId: file ID, globally unique
- DbName: Name of database to which it belongs
- Catalog: Custom classification
- FileName: File name
- FileSize: File size, in bytes
- MD5: MD5 of files.

Example

1. View the uploaded files in the database my_database

```
SHOW FILE FROM my_database;
```

Keywords

```
SHOW,FILE
```

SHOW-ROLES

SHOW ROLES Description

This statement is used to show all created role information, including name of roles, users and privileges contained.

```
SHOW ROLES;
```

Description of return results:

```
mysql> show roles;
+---------+------------+---------------------+--------------+-----------+
| Name    | Users      | GlobalPrivs         | DatabasePrivs | TablePrivs |
+---------+------------+---------------------+--------------+-----------+
| admin   | 'admin'@'%' | Admin_priv          | N/A          | N/A       |
+---------+------------+---------------------+--------------+-----------+
2 rows in set (0.00 sec)
```

- `Name`

  Name of roles.

- `Users`

  List of `user_identity` belonging to this role.

- `GlobalPrivs`

  Global privilege list.

- `DatabasePrivs`

  List of privileges at the database level.

- `TablePrivs`

  List of privileges at table level.

Example

1. View roles created:

```
SHOW ROLES;
```

Keywords

```
SHOW, ROLES
```

SHOW-GRANTS

SHOW GRANTS Description

This statement is used to view the grants for users.

```
SHOW [ALL] GRANTS [FOR user_identity];
```

- ALL

  By using the key word ALL, user can view the grants for all users.

2. user_identity

   View the grants for specified user. And the user_identity must be created by CREATE USER command.

   If user_identity is not specified, then view the grants for current user.

Description of return results:

```
mysql> show grants;
+------------+---------+-------------------+--------------+-----------+--------------+
| UserIdentity | Password | GlobalPrivs          | DatabasePrivs | TablePrivs | ResourcePrivs |
+------------+---------+-------------------+--------------+-----------+--------------+
| 'admin'@'%'  | Yes      | Admin_priv (false) | NULL          | NULL       | NULL          |
+------------+---------+-------------------+--------------+-----------+--------------+
1 row in set (0.00 sec)
```

- UserIdentity

  User identity.

- Password

  Yes indicates that a password is set. No indicates no password.

- GlobalPrivs

  List of grants at the global level. (false after list can be ignored).

- DatabasePrivs

  List of grants at the database level. (false after list can be ignored).

- TablePrivs

  List of grants at the table level. (false after list can be ignored).

Example

1. View the grants for all users

```
SHOW ALL GRANTS;
```

2. View the grants for specified users

```
SHOW GRANTS FOR jack@'%';
```

3. View the grants for current user

```
SHOW GRANTS;
```

Keywords

```
SHOW, GRANTS
```

SHOW-EXPORT

SHOW EXPORT Description

This statement is used to show the execution of specified export tasks.

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

- db_name : View jobs under the specified database. If the database is not specified, then use the current database.
- WHERE : results can be filtered by EXPORT_JOB_ID or STATE .
- ORDER BY LIMIT : support ordering and paging result sets by any column.

Description of return results:

```
     JobId: 14008
     State: FINISHED
  Progress: 100%
  TaskInfo: {"partitions":["*"],"exec mem limit":2147483648,"column separator":",","line delimiter":"\n","tablet
num":1,"broker":"hdfs","coord num":1,"db":"default_cluster:db1","tbl":"tbl3"}
      Path: bos://bj-test-cmy/export/
CreateTime: 2019-06-25 17:08:24
 StartTime: 2019-06-25 17:08:28
FinishTime: 2019-06-25 17:08:34
   Timeout: 3600
  ErrorMsg: N/A
```

- JobId: unique ID of job
- State: Job status:
  - PENDING : job pending for scheduled
  - EXPORTING : exporting data
  - FINISHED : job finished successfully
  - CANCELLED : job failed
- Progress : job progress. The progress takes query plan as unit. Assuming that there are 10 query plans in total and 3 of which have been completed at present, then the progress is 30%.
- TaskInfo : Job information shown in Json format:
  - db: Database name
  - tbl: Table name
  - partitions: specify the exported partition.* * represents all partitions.
  - exec mem limit: query the planned use limit of memory, in bytes.
  - column separator: column separator of export file.
  - line delimiter: line separator of export file.
  - tablet num: total number of Tablet involved.
  - broker: name of broker used.

    - coord num: number of query plans.
- Path: export path on remote storage.
- CreateTime/StartTime/FinishTime: job create time, start time and finish time.
- Timeout: job timeout period, in seconds, starting from Create Time.
- ErrorMsg: if there is an error in the job, the error reason will be displayed here.

Example

1. Show all export tasks under default db

```
SHOW EXPORT;
```

2. Show the export tasks under specified db, order in descending order of StartTime

```
SHOW EXPORT FROM example_db ORDER BY StartTime DESC;
```

3. Show the export task under specified db, the state is "exporting", and order in descending order of StartTime

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

4. Show the export task under specified db and specified job_id

```
SHOW EXPORT FROM example_db WHERE EXPORT_JOB_ID = job_id;
```

Keywords

```
SHOW, EXPORT
```

SHOW-ROUTINE-LOAD-TASK

SHOW ROUTINE LOAD TASK

Description

View the subtasks of a specified Routine Load job currently running .

```
SHOW ROUTINE LOAD TASK
WHERE JobName = "job_name";
```

The return result is as follows:

```
              TaskId: d67ce537f1be4b86-abf47530b79ab8e6
               TxnId: 4
           TxnStatus: UNKNOWN
               JobId: 10280
          CreateTime: 2020-12-12 20:29:48
   ExecuteStartTime: 2020-12-12 20:29:48
             Timeout: 20
                BeId: 10002
DataSourceProperties: {"0":19}
```

- `TaskId` : Unique ID of subtask.
- `TxnId` : Load job ID corresponding to subtask.
- `TxnStatus` : Load job status corresponding to subtask, generally UNKNOWN, with no practical meaning.
- `JobId` : Job ID corresponding to the subtask.
- `CreateTime` : The time wehn the subtask was created.

- `ExecuteStartTime` : the time when the subtask is scheduled for execution, generally later than the create time.
- `Timeout` : Timeout period of subtasks, usually two times of `MaxIntervalS` set in job.
- `BeId` : ID of BE node this subtask is running on.
- `DataSourceProperties` : Start offset of Kafka Partition that to be consumed by subtask. It is a string in Json format. Key is Partition Id. Value is the start offset consumed.

Example

1. Show subtask information of routine load task named test1.

```
SHOW ROUTINE LOAD TASK WHERE JobName = "job";
```

Keywords

```
SHOW, ROUTINE, LOAD, TASK
```

Best Practices

The number of subtasks of a Routine Load job currently running and the specific BE node they are running on can be viewed by using this command.

SHOW-REPOSITORIES

SHOW REPOSITORIES Description

This statement is used to view the repositories created.

```
SHOW REPOSITORIES;
```

Description of return results:

```
    RepoId: 385148
  RepoName: bos_repo
CreateTime: 2020-09-28 19:46:00
IsReadOnly: false
  Location: bos://my_bucket/palo_backup
    Broker: bos
    ErrMsg: NULL
```

- `RepoId` : Unique ID of each repository.
- `RepoName` : name of repository.
- `CreateTime` : The time when the respository was first created.
- `IsReadOnly` : Whether it is a read-only respository.
- `Location` : Root directory in the respository for backing up data.
- `Broker` : Name of Broker service relied on.
- `ErrMsg` : Palo will regularly check the connectivity of repositories, and if there is any problem found, an error message will be displayed here.

Example

1. View the repositories created:

```
SHOW REPOSITORIES;
```

Keywords

```
SHOW, REPOSITORY, REPOSITORIES
```

SHOW-RESOURCES

SHOW RESOURCES

Description

This statement is used to show the resources that the user has privilege to use. For ordinary users, only resources that the

ordinary users with privilege to use will be displayed. While for admin users, all resources will be displayed.

```
SHOW RESOURCES
[
    WHERE
    [NAME [ = "your_resource_name" | LIKE "name_matcher"]]
    [RESOURCETYPE = ["ODBC"]]
]
[ORDER BY ...]
[LIMIT limit][OFFSET offset];
```

- `WHERE`

  It supports filtering resources by where condition. It can use `NAME` for exact matching of equivalent value or use Like for fuzzy matching. `RESOURCETYPE` can be used to match resource types.

- `ORDER BY LIMIT`

  It supports ordering and paging result sets by any column.

Description of return results:

```
mysql> show resources;
+------------------+--------------+----------+--------------+
| Name             | ResourceType | Item     | Value        |
+------------------+--------------+----------+--------------+
| my_reousrce      | odbc_catalog | password |              |
| my_reousrce      | odbc_catalog | database | test         |
| my_reousrce      | odbc_catalog | driver   | MySQL        |
| my_reousrce      | odbc_catalog | port     | 9030         |
| my_reousrce      | odbc_catalog | odbc_type | mysql       |
| my_reousrce      | odbc_catalog | host     | 127.0.0.1    |
| my_reousrce      | odbc_catalog | type     | odbc_catalog |
| my_reousrce      | odbc_catalog | user     | admin        |
+------------------+--------------+----------+--------------+
```

The properties of a resource will be displayed in multiple lines, one property in each line.

- `Name` : resource name.
- `ResourceType` : resource type.

- `Item` : property key.
- `Value` : property value.

> Note: the password will not be displayed.

Example

1. Show resources created.

```
SHOW RESOURCES;
```

2. Filter and order resources according to conditions.

```
SHOW RESOURCES
WHHERE name = "my_reousrce"
ORDER BY Name DESC
LIMIT 0, 10;
```

Keywords

```
SHOW, RESOURCES
```

SHOW-LOAD

SHOW LOAD Description

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

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

- db_name

  View load jobs under specified database. If not specified, use the current database.

- LABEL

  LLabel can be accurately matched by equivalent value, or fuzzy matched by Like.

- STATE

  View load tasks in specified status.

  1. PENDING: load has been submitted, but has not been executed yet.
  2. LOADING: load is in progress.
  3. FINISHED: load is finished successfully.
  4. CANCELLED: load failed.

- ORDER BY ... LIMIT [OFFSET]

  It supports ordering result sets by any column.

  ORDER BY createtime DESC LIMIT 10,20

The return result of the SHOW LOAD command is as follows:

```
 mysql> show load order by createtime desc limit 1\G
*************************** 1. row ***************************
        JobId: 76391
        Label: label1
        State: FINISHED
     Progress: ETL:N/A; LOAD:100%
         Type: BROKER
      EtlInfo: unselected.rows=4; dpp.abnorm.ALL=15; dpp.norm.ALL=28133376
     TaskInfo: cluster:N/A; timeout(s):10800; max_filter_ratio:5.0E-5
     ErrorMsg: N/A
   CreateTime: 2019-07-27 11:46:42
 EtlStartTime: 2019-07-27 11:46:44
EtlFinishTime: 2019-07-27 11:46:44
LoadStartTime: 2019-07-27 11:46:44
LoadFinishTime: 2019-07-27 11:50:16
          URL: http://192.168.1.1:8040/api/_load_error_log?file=__shard_4/error_log_insert_stmt_4bb00753932c491a-a6da6e2725415317_4bb00753932c491a_a6da6e2725415317
   JobDetails: {"Unfinished backends":{"9c3441027ff948a0-8287923329a2b6a7":[10002]},"ScannedRows":2390016,"TaskNumber":1,"All backends":{"9c3441027ff948a0-8287923329a2b6a7":[10002]},"FileNumber":1,"FileSize":1073741824}
```

The follow mainly introduces that meaning of parameters in the result set return by view load command:

- JobId

  unique ID of load task, JobId in different load tasks is different, which is automatically generated by the system. Unlike Label, JobId will never be the same, and Label can be duplicated after the load task fails.

- Label

  Identification of load task.

- State

  Current stage of the load task. There will be PENDING and LOADING status appearing in the loading process of Broker. If Broker load is in PENDING status, it means that the current load task is waiting to be executed. If in LOADING status, it indicates that it is being executed.

  There are two final stages of the load task: CANCELLED and FINISHED, when the Load job is in these two stages, the load is completed. In which CANCELLED means load failed and FINISHED means load succeeded.

- Progress

  Description of load task progress. There are two kinds of progress: ETL and LOAD, corresponding to the two stages of the load process: ETL and LOADING. At present, Broker load has only LOADING stage, so ETL will always be displayed as N/A

  The progress range of LOAD is: 0~100%。

  ```
  LOAD progress = the number of tables currently loaded/the total number of tables designed for this load task * 100%
  ```

  If all load tables are loaded, then the progress of LOAD will show 99% and the load will enter the final effective stage. After the whole load is completed, the progress of LOAD will be changed to 100%.

  The progress of loading is not linear. Therefore, if the progress does not change within a period of time, it does not mean that the load is not being executed.

- Type

  Type of task loaded. Broker load has only the type BROKER.

- EtlInfo

  It mainly displays the loaded data quantity indicators unselected.rows , dpp.norm.ALL and dpp.abnorm.ALL. Users can

judge how many rows are filtered by the where condition according to the first numerical value, and verify whether the error rate of the current load task exceeds `max_filter_ratio` by the latter two indicators.

The sum of three indicators is the total row number of the original data.

- `TaskInfo`

  It mainly displays the current load task parameters, that is, the load task parameters specified by the user when creating the Broker load task, including:`cluster` , `timeout` and`max_filter_ratio`.

- `ErrorMsg`

  When the load task status is CANCELLED, the reason of failure will be displayed in two parts: type and msg, when the load task is successful, display `N/A`.

  The value meaning of type:

  ```
  USER_CANCEL:task cancelled by user
  ETL_RUN_FAIL:load task failed in ETL phase
  ETL_QUALITY_UNSATISFIED:data quality is unqualified, that is, the error data rate exceeds max_filter_ratio
  LOAD_RUN_FAIL: load task TIMEOUT failed in LOADING phase:
  TIMEOUT : load task did not complete within timeout period
  UNKNOWN: unknown load error
  ```

- `CreateTime/EtlStartTime/EtlFinishTime/LoadStartTime/LoadFinishTime`

  These values represent the load create time, ETL stage start time, ETL stage finish time, loading stage start time and the whole load task finish time.

  There is no ETL stage in Broker load, so its ETL starttime, ETL finishtime and loadstarttime are set to the same value.

  When the load task stays at CreateTime for a long time, while LoadStartTime is N/A, it indicates that load task is seriously piled up at present. Users can reduce the frequency of load submission.

  ```
  LoadFinishTime - CreateTime = time consumed by the whole load task
  LoadFinishTime - LoadStartTime = Execution time of the whole Broker load task  = Time consumed by the whole load task-Waiting time of the load task
  ```

- `URL`

  Load the error data sample of the task, and access the URL address to get the error data sample loaded this time. If there is no error data in this load, the URL field is N/A.

- `JobDetails`

  Display the detailed running status of some jobs, including the number of loaded files, the total size (bytes), the number of subtasks, the number of processed original rows, the BE node Id of running subtasks, and the BE node Id of unfinished subtasks.

  ```
  {"Unfinished backends":{"9c3441027ff948a0-8287923329a2b6a7":
  [10002]},"ScannedRows":2390016,"TaskNumber":1,"All backends":{"9c3441027ff948a0-8287923329a2b6a7":
  [10002]},"FileNumber":1,"FileSize":1073741824}
  ```

  In which the number of processed original rows is updated every 5 seconds. The number of rows is only used to show the current progress, which does not represent the final actual number of rows processed. The actual number of rows processed shall be as shown in EtlInfo.

Example

1. Show all load tasks of the default db

```
SHOW LOAD;
```

2. Show the load task of specified db, the label contains the string "2014_01_02".

```
SHOW LOAD FROM example_db WHERE LABEL LIKE "2020_01_02" LIMIT 10;
```

3. 展示指定 db 的导入任务，指定 label 为 "load_example_db_20140102" 并按 LoadStartTime 降序排序。

```
SHOW LOAD FROM example_db WHERE LABEL = "load_example_db_20200102" ORDER BY LoadStartTime DESC;
```

4. Show the load task of the specified db, specify the label as "load_example_db_20140102" and order in descending order of LoadStartTime

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

5. 展示指定 db 的导入任务 并按 LoadStartTime 降序排序,并从偏移量5开始显示10条查询结果

```
SHOW LOAD FROM example_db ORDER BY LoadStartTime DESC limit 5,10;
SHOW LOAD FROM example_db ORDER BY LoadStartTime DESC limit 10 offset 5;
```

Keywords

```
SHOW, LOAD
```

Best Practices

1. Use in combination with LOAD command

   It is recommended that user view the status of load tasks submitted by polling by `SHOW LOAD` . For example, for an load task with a specified Label, the load status is polled at intervals of 10 seconds or longer, and the next operation is performed according to the status.

   For example, wait for the load task submitted in the previous batch to be completed before submitting in the next batch, or retry the failed task.

2. Query multiple load tasks with the same Label.

   If the load task corresponding to a Label fails, the user can continue to use the Label to resubmit the task. In this case, querying the Label will return multiple load jobs (including newly submitted and previously failed). At this time, if to query the latest submission, you can use the following command:

   ```
   SHOW LOAD WHERE LABEL="my_label" ORDER BY createtime DESC LIMIT 1;
   ```

3. View data quality problem

   If the load task fails and the error message shows `ETL_QUALITY_UNSATISFIED` , it indicates that there is load quality problem. A link will be given in `URL` field. The error data information given in this link can be viewed with the following command:

```
SHOW LOAD WARNINGS ON "your_url";
```

Here are some common data quality problems:

- `no partition for this tuple`

  This row of data has no corresponding partition in the table. It is required to check whether the data falls within the partition range of the table.

- `null is not allowed for bitmap column`

  Loading null values into bitmap type field is not allowed.

- `the length of input is too long than schema`

  The length of string exceeds the length defined by the column. It is required to consider increasing the defined maximum length of VARCHAR column.

- `decimal value is not valid for definition`

  Decimal type does not match in precision.

- `Content of HLL type column is invalid`

  The value type corresponding to HLL column is not correct. It is required to check whether the hll_hash function is used to convert the data.

- `null value for not null column`

  Try to load a null value into a column that is not nullable

- `Parse json data for JsonDoc failed`

  Parsing Json data failed.

- `JSON Root not found`

  Given json_root does not match.

- `JSON data is array-object, strip_outer_array must be TRUE.`

  `strip_outer_array` is false, but the Json data is an array.

- `JSON data is not an array-object, strip_outer_array must be FALSE`

  `strip_outer_array` is true, but the Json data is not an array.

- `Json value is null, but the column xx is not nullable`

  While loading Json data, try to load null into a field that is not nullable.

- `The column xx is not nullable, but it's not found in jsondata.`

  When loading Json data, no field matched, and the coloumn is not nullable.

- `All fields is null, this is a invalid row.`

  When loading Json data, all field names in a row cannot match the data.

- `Empty json line`

Empty Json data.

- `data is not encoded by UTF-8`

  Source data is not UTF-8 encoded.

- `actual column number is less than schema column number.`

  The number of columns in the source data is less than that in the table.

- `actual column number is more than schema column number.`

  The number of columns in the source data is more than that in the table.

- `column xx value is incorrect while strict mode is true`

  In Strict Mode , the source data is converted to null.

- `column xx value is null while columns is not nullable`

  Try to load null value into the coloumn is not nullable.

4. View the progress of job execution

   As the progress displayed in `Progress` in the returned result is not linear, sometimes it is inconvenient to query progress through this field. At this time, we can also observe whether the job is running through `ScannedRows` in `JobDetails` field.

   Under normal circumstances, the `ScannedRows` of a running job will be updated every 5 seconds, indicating the number of rows of data that have been read. But sometimes, due to the data quality problems, all data rows have been filtered, resulting in the `ScannedRows` never increasing or being 0. At this time, `All backends` and `Unfinished backends` fields can also be viewed. The former represents all BE nodes involved in this load task, while the latter represents the BE nodes of unfinished tasks.

5. About TaskNumber

   The TaskNumber field in `JobDetails` does not represent the concurrency of load jobs. This only represents the number of `DATA INFILE` clauses in the Load statement.

SHOW-RESTORE

SHOW RESTORE Description

This statement is used to view the RESTORE task

```
SHOW RESTORE [FROM db_name];
```

Description of return results

- `JobId`: unique job id.
- `Label`: the name of the backup to be restored.
- `Timestamp`: time version of the backup to be restored.
- `DbName`: database belonged to.

- State: current stage.
  - PENDING: initial state after job submission.
  - SNAPSHOTING: executing snapshot.
  - DOWNLOAD: snapshot completed, ready to download the snapshot in the respository.
  - DOWNLOADING: snapshot is being downloaded.
  - COMMIT: snapshot download completed, ready to take effect.
  - COMMITING: in effect.
  - FINISHED: job finished successfully.
  - CANCELLED: job failed.
- AllowLoad: whether to allow load during recovery (currently not supported).
- ReplicationNum: specify the number of replication to recover.
- RestoreJobs : tables and partitions to recover.
- CreateTime : task submission time.
- MetaPreparedTime : completion time of metadata preparation.
- SnapshotFinishedTime : snapshot finished time.
- DownloadFinishedTime : snapshot downloading finished time.
- FinishedTime : job finished time.
- UnfinishedTasks : unfinished subtask id will be displayed in snapshot, download, and COMMITING phases.
- Status : if the job fails, show the failure information.
- Timeout : timeout period of job, in seconds.

Example

1. View the latest RESTORE task under example_db.

```
SHOW RESTORE FROM example_db;
```

keywords

```
SHOW, RESTORE
```

SHOW PROPERTY Description

This statement is used to view user properties

```
SHOW PROPERTY [FOR user] [LIKE key];
```

- `user`

  View the properties of specified user. If not specified, view the properties of current user.

- `LIKE`

  Fuzzy matching can be performed by name of properties.

Description of return results:

```
mysql> snow property like '%connection%';
+----------------------+-------+
| Key                  | Value |
+----------------------+-------+
| max_user_connections | 100   |
+----------------------+-------+
1 row in set (0.01 sec)
```

- `Key`

  Property name.

- `Value`

  Property value.

Example

1. View the properties of jack user

```
SHOW PROPERTY FOR 'jack';
```

2. View the property of jack user like connection limit

```
SHOW PROPERTY FOR 'jack' LIKE '%connection%';
```

Keywords

```
SHOW, PROPERTY
```

SHOW-BACKUP

SHOW BACKUP Description

This statement is used to view BACKUP tasks

```
SHOW BACKUP [FROM db_name];
```

Description of return results:

```
 mysql> show backup\G
*************************** 1. row ***************************
            JobId: 89205228
     SnapshotName: snapshot1
           DbName: example_db
            State: FINISHED
       BackupObjs: [default_cluster:example_db.table1 (p1)]
       CreateTime: 2020-06-14 12:08:49
SnapshotFinishedTime: 2020-06-14 12:08:55
 UploadFinishedTime: 2020-06-14 12:10:31
     FinishedTime: 2020-06-14 12:10:41
   UnfinishedTasks:
         Progress:
        TaskErrMsg:
           Status: [OK]
          Timeout: 86400
1 row in set (0.06 sec)
```

1. Palo only saves the latest BACKUP task.

2. The meaning of each column is as follows:

- `JobId`：unique job id.
- `SnapshotName`: name of the backup.
- `DbName`：database belonged to
- `State`：current stage
  - `PENDING`：initial state after job submission.
  - `SNAPSHOTING`：executing snapshot.
  - `UPLOAD_SNAPSHOT`：snapshot completed, ready to upload.
  - `UPLOADING`：uploading snapshot.
  - `SAVE_META`：save job meta information as a local file.
  - `UPLOAD_INFO`：upload job meta information.
  - `FINISHED`：job finished successfully.
  - `CANCELLED`：job failed.
- `BackupObjs`: tables and partitions of backup.
- `CreateTime`：task submission time.
- `SnapshotFinishedTime`：task submission time.
- `UploadFinishedTime`：Snapshot uploading finished time.
- `FinishedTime`：job finished time.
- `UnfinishedTasks`：show unfinished sub-tasks id in SNAPSHOTING and UPLOADING stages.
- `Status`：If the job fails, show the failure information.
- `Timeout`：Timeout period of job, in seconds.

example

1. View the latest BACKUP task under `example_db` .

```
SHOW BACKUP FROM example_db;
```

Keywords

```
SHOW, BACKUP
```

最佳实践

1. Only show the information of the latest backup operation. Historical operation information cannot be viewed.

SHOW-ALTER-TABLE-MATERIALIZED-VIEW

SHOW ALTER TABLE MATERIALIZED VIEW

Description

This command is used to view the execution of creating materialized views submitted by CREATE-MATERIALIZED-VIEW statement.

> This statement is equivalent to `SHOW ALTER TABLE ROLLUP;`

```
SHOW ALTER TABLE MATERIALIZED VIEW
[FROM database]
[WHERE]
[ORDER BY]
[LIMIT OFFSET]
```

- database: view jobs under specified database. If not specified, use the current database.
- WHERE: support filtering the results, currently only the following columns can be filtered:
  - TableName: only equivalent filtering is supported.
  - State：Only equivalent filtering is supported.
  - Createtime/FinishTime: support =, >=, <=, >, <, !=
- ORDER BY: support ordering result sets by any column.
- LIMIT: support page turning and query in combination with ORDER BY.

Description of return results:

```
 mysql> show alter table materialized view\G
*************************** 1. row ***************************
          JobId: 11001
      TableName: tbl1
     CreateTime: 2020-12-23 10:41:00
     FinishTime: NULL
  BaseIndexName: tbl1
RollupIndexName: r1
       RollupId: 11002
  TransactionId: 5070
          State: WAITING_TXN
            Msg:
       Progress: NULL
        Timeout: 86400
1 row in set (0.00 sec)
```

- `JobId` : Unique ID of a job.
- `TableName` : Base table name
- `CreateTime/FinishTime` : job create time and finish time.
- `BaseIndexName/RollupIndexName` : Base table name and materialized view name.
- `RollupId` : Unique ID of materialized view.
- `TransactionId` : See description of State field.
- `State` : Job status.
  - PENDING : preparing for job.
  - WAITING_TXN :

    Before officially generating materialized view data, wait for the running load job on the current table to finish. And the`TransactionId` field is the ID of the job currently waiting for. When the load jobs before this ID have been completed, the job will start actually.
  - RUNNING : job in running.
  - FINISHED : job finished successfully.
  - CANCELLED : job failed.
- `Msg`: error message
- `Progress` : job progress. The progress here is expressed as `Number of tablet completed/total number of tablet`. The materialized views are created by the granularity of tablet.
- `Timeout` : timeout period of job, in seconds.

Example

1. View materialized view jobs under database example_db

```
SHOW ALTER TABLE MATERIALIZED VIEW FROM example_db;
```

Keywords

```
SHOW, ALTER, TABLE, MATERIALIZED, VIEW
```

SHOW-SNAPSHOT

SHOW SNAPSHOT Description

This statement is used to view backup snapshots that already exist in the respository.

```
SHOW SNAPSHOT ON `repo_name`
[WHERE SNAPSHOT = "snapshot" [AND TIMESTAMP = "backup_timestamp"]];
```

The results can be filtered by using `WHERE` statement, and more detailed information about a snapshot can be displayed.

Description of return results:

1. View all snapshots:

```
mysql> show snapshot on bos_repo;
+-----------+--------------------+--------+
| Snapshot  | Timestamp          | Status |
+-----------+--------------------+--------+
| snapshot1 | 2020-06-11-20-56-29 | OK     |
| snapshot2 | 2020-06-11-15-34-35 | OK     |
| snapshot3 | 2020-06-11-20-39-20 | OK     |
+-----------+--------------------+--------+
```

- `Snapshot` : name of backup.
- `Timestamp` : time version of corresponding backup.
- `Status` : If the backup is normal, it will show OK; otherwise, it will show an error message.

2. View the time version of specified snapshot:

```
mysql> snow snapshot on bos_repo where snapshot = "snapshot1" and timestamp="2020-06-11-20-56-29"\G
*************************** 1. row ***************************
 Snapshot: snapshot1
Timestamp: 2020-06-11-20-56-29
 Database: example_db
  Details: {
            "database": "example_db",
            "meta_version": 93,
            "backup_time": 1591880189,
            "name": "snapshot1",
            "backup_result": "succeed",
            "backup_objects": {
             "table1": {"partitions": {"table1": {}}},
             "table1": {"partitions": {"p1": {}}}
            }
           }
   Status: OK
```

If TIMESTAMP is specified, the following additional information will be displayed:

- `Database` : name of the database in the backup data source.
- `Details` : Show the entire backup data directory and file structure in the form of Json.

Example

1. View the existing backups in repositoryexample_repo:

```
SHOW SNAPSHOT ON example_repo;
```

2. Only view the backups named backup1 in repository example_repo:

```
SHOW SNAPSHOT ON example_repo WHERE SNAPSHOT = "backup1";
```

3. View the details of the backup named backup1 in the repository example_repo with the time version of "2018-05-05-15-34-26":

```
SHOW SNAPSHOT ON example_repo
WHERE SNAPSHOT = "backup1" AND TIMESTAMP = "2020-05-05-15-34-26";
```

Keywords

```
SHOW, SNAPSHOT
```

SHOW-ROUTINE-LOAD

SHOW ROUTINE LOAD

Description

Used to display Routine Load job information.

```
SHOW [ALL] ROUTINE LOAD
[FOR [db.]job_name]
```

- ALL : If ALL keyword is specified, then all running and stopped jobs will be displayed. Otherwise, only jobs that are not STOPPED are displayed.
- db : If no database is specified, the jobs under the current database will be displayed by default.

Return results of statements are as follows:

```
 mysql> show routine load\G
*************************** 1. row ***************************
                  Id: 10280
                Name: job1
          CreateTime: 2020-12-10 19:32:58
           PauseTime: NULL
             EndTime: NULL
              DbName: default_cluster:db1
           TableName: tbl1
               State: RUNNING
      DataSourceType: KAFKA
      CurrentTaskNum: 1
       JobProperties:
{"partitions":"*","columnToColumnExpr":"","maxBatchIntervalS":"10","whereExpr":"*","timezone":"Asia/Shanghai","merge
}
DataSourceProperties: {"topic":"test","currentKafkaPartitions":"0","brokerList":"127.0.0.1:9094"}
    CustomProperties: {}
           Statistic:
{"receivedBytes":0,"errorRows":0,"committedTaskNum":0,"loadedRows":0,"loadRowsRate":0,"abortedTaskNum":0,"totalRo
}
           Progress: {"0":"OFFSET_BEGINNING"}
ReasonOfStateChanged:
        ErrorLogUrls:
             OtherMsg:
1 row in set (0.01 sec)
```

- `Id`: unique ID of job.
- `Name`: job name.
- `CreateTime`: job create time.
- `PauseTime` : the time when the last job was suspended.
- `EndTime` : job finished time.
- `DbName` : database name.
- `TableName`: name of table loaded.
- `State` : running status of job.
  - `NEED_SCHEDULE` : job pending for scheduling
  - `RUNNING` : job is running.
  - `PAUSED` : job is paused.
  - `STOPPED` : job has been stopped.
- `DataSourceType` : data source type.

- `CurrentTaskNum` : number of subtasks running.
- `JobProperties` : Job configuration details, a string in Json format.

```
{
    "partitions": "*",                  // Specify the loaded partition list, * represents unspecified.
    "columnToColumnExpr": "",           // Column mapping and conversion relationships.Empty means unspecified.
    "maxBatchIntervalS": "10",          // Maximum running time of subtasks, in seconds.
    "whereExpr": "*",                   // Column filter criteria.* represents unspecified.
    "timezone": "Asia/Shanghai",        // time zone:
    "mergeType": "APPEND",              // data merger type.
    "format": "csv",                    // load data format.
    "columnSeparator": "','",           // column separator.
    "json_root": "",                    // Json Root。
    "maxBatchSizeBytes": "104857600",   // maximum bytes consumed by subtasks.
    "exec_mem_limit": "2147483648",     // memory limits for subtasks.
    "strict_mode": "false",             // whether to turn on strict mode.
    "jsonpaths": "",                    // json paths
    "deleteCondition": "*",             // whether the Marked Delete column is specified. * represents unspecified.
    "desireTaskConcurrentNum": "1",     // the expected maximum number of concurrent subtasks set by the user.
    "maxErrorNum": "0",                 // max rows with errors allowed.
    "strip_outer_array": "false",       // Whether to expand the array for Json format data.
    "currentTaskConcurrentNum": "1",    // Current number of concurrent subtasks.
    "num_as_string": "false",           // Whether to parse all fields in Json data into string type.
    "maxBatchRows": "200000"            // Maximum row number consumed by subtasks.
}
```

- `CustomProperties` : extra property configured by users. It is a string in Json format.
- `Statistic` : statistical information of job running. It is a string in Json format.

```
{
    "receivedBytes": 0,         // bytes of received data.
    "errorRows": 0,             // error data rows.
    "committedTaskNum": 0,      // number of subtasks successfully loaded.
    "loadedRows": 0,            // number of rows loaded.
    "loadRowsRate": 0,          // average number of rows loaded per second.
    "abortedTaskNum": 0,        // number of subtasks that failed or did not consume data.
    "totalRows": 0,             // total number of rows consumed.
    "unselectedRows": 0,        // number of rows filtered by the where condition.
    "receivedBytesRate": 0,     // bytes of data received per second.
    "taskExecuteTimeMs": 1      // accumulated execution time of subtasks.
```

```
}
```

- `ReasonOfStateChanged` : the reason for the change of job status.
- `ErrorLogUrls` : When there is error data, the url links of error data in last three subtasks with error data will be displayed here. The error data can be viewed through the following statement.

```
SHOW LOAD WARNINGS ON "your_url";
```

Description of error messages can be found in SHOW LOAD

- `Progress` : consumption progress. It is a string in Json format.Key is the Partition ID of Kafka. The meaning of Value is as follows:
  - OFFSET_BEGINNING : initial state, which means that the consumption starts from the beginning and has not started yet.
  - OFFSET_END : initial state, which means that the consumption starts from the end and has not started yet.
  - OFFSET_ZERO : initial state, which means that the consumption starts from 0 and has not started yet.
  - Integer value: offset that **has been consumed** by corresponding Kafka partition.
- `OtherMsg`: other information.

example

1. Show all routine load jobs named test1 (including stopped or cancelled jobs). The result is in one or several rows.

```
SHOW ALL ROUTINE LOAD FOR test1;
```

2. Show the currently running routine load job named test1

```
SHOW ROUTINE LOAD FOR test1;
```

3. Show all routine load jobs (including stopped or cancelled jobs) under example_db. The result is in one or several rows.

```
use example_db;
SHOW ALL ROUTINE LOAD;
```

4. Show all running routine load jobs under example_db

```
use example_db;
SHOW ROUTINE LOAD;
```

5. Show the currently running routine load job named test1 under example_db

```
SHOW ROUTINE LOAD FOR example_db.test1;
```

6. Show all routine load jobs named test1 under example_db (including stopped or cancelled jobs). The result is in one or several rows.

```
SHOW ALL ROUTINE LOAD FOR example_db.test1;
```

Keywords

```
SHOW, ROUTINE, LOAD
```

Best Practices

1. Observe the progress of load jobs.

   The consumption progress of the load job can be viewed through the following fields in the return results.

   1. `Statistic`

      Observe the changes of `committedTaskNum` and `abortedTaskNum` in Statistic. If `committedTaskNum` continues to increase, it indicates that the job is consuming normally. If `abortedTaskNum` continues to increase, it indicates that there might be no data to be consumed in Kafka; or the data in Kafka cannot be consumed; or there are other errrors encountered.

   2. `Progress`

      Observe the offset consumption progress of Kafka Partition to know to overal consumption progress.

Account Management

SET-PROPERTY

SET PROPERTY description

```
SET PROPERTY [FOR 'user']
'key' = 'value' [, 'key' = 'value'];
```

Set user property, such as number of connections etc..

The user property set here is special for `user` but not for `user_identity`.

That is, assuming that two users have been created by using `CREATE USER` statement, namely user `'jack'@'%'` and user `'jack'@'192.%'`, then use `SET PROPERTY` statement, which is special for `jack` user, but not for `'jack'@'%'` user or `'jack'@'192.%'` user.

- `key`

Currently it supports the following properties:

- max_user_connections

  Max number of connections allowed for the user.

Example

1. Modify the max number of connections for user jack as 1000

```
SET PROPERTY FOR 'jack' 'max_user_connections' = '200';
```

Keywords

```
SET, PROPERTY
```

Best Practices

1. Max number of connections.

   The max number of connections refers to the limit on max connections of the user on one Leader Node, which defaults to 100. If a cluster has 3 Leader Nodes, then the max number of connections is 300 theoretically.

   The Palo cluster has upper limit on total number of connections, which defaults to 1024. That mean, the number of connections of all users should not exceed this upper limit.

   If it is a high concurrency application, it is recommended to connect and duplicate these connections at business side through thread pool. Generally the default limit on number of connections can satisfy the access scenario of high concurrency by thousands of QPSs.

SET-PASSWORD

SET PASSWORD Description

Used to modify the login password of a user.

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

- "user_identity"

  If not specified, then modify the password of current user.

  Note the user_identity here must completely match with the user_identity specified when creating user by using CREATE USER , otherwise error will be reported that the user does not exist. If the user_identity is not specified, the current user is 'username'@'ip'. And the current user may not match with any user_identity. The user_identity corresponding to current user can be viewed by command :

```
select current_user();
```

- PASSWORD

  The password input byPASSWOPD() is cleartext.

  And the password that uses character string directly and should be transferred is encrypted password.

To modify the password of other uses, the priviledge of admin is required.

Example

1. Modify the password of current user

```
SET PASSWORD = PASSWORD('123456');

SET PASSWORD = '*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9';
```

2. Modify the password of specified user

```
SET PASSWORD FOR 'jack'@'192.%' = PASSWORD('123456');

SET PASSWORD FOR 'jack'@['domain'] = '*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9';
```

Keywords

```
SET, PASSWORD
```

DROP-USER

DROP USER Description

This statement is used to drop a created user.

```
DROP USER 'user_identity'
```

- user_identity

  User name to be dropped.

```
 user@'host'
user@['domain']
```

Example

1. Drop user jack@'192.%'

```
DROP USER 'jack'@'192.%'
```

Keywords

```
DROP, USER
```

CREATE-USER

CREATE USER

Description

This statement is used to create a user. The common account can be used to log in and operate Palo later.

```
CREATE USER user_identity
[IDENTIFIED BY 'password']
[DEFAULT ROLE 'role_name']
```

- user_identity

  User identity.

```
'user_name'@'host'
```

- passwrod

  Password. Optional, default to empty.

- role

  Role. If specified, user will have the grant to change roles.

Example

1. Create a user with no password set (if host is not specified, then equivalent to jack@'%')

```
CREATE USER 'jack';
```

2. Create a user set with password, allow login from' 172.10.1.10'

```
CREATE USER jack@'172.10.1.10' IDENTIFIED BY '123456';
```

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

```
CREATE USER jack@'172.10.1.10' IDENTIFIED BY PASSWORD
'*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9';
```

The encrypted content can be obtained by the `PASSWORD` function later, for example:

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

4. Create a user who is allowed to log in from the' 192.168' subnet and specify its role as example_role

```
CREATE USER 'jack'@'192.168.%' DEFAULT ROLE 'example_role';
```

5. Create a user who is allowed to log in from the domain name' example_domain'

```
CREATE USER 'jack'@['example_domain'] IDENTIFIED BY '12345';
```

6. Create a user and specify a role

```
CREATE USER 'jack'@'%' IDENTIFIED BY '12345' DEFAULT ROLE 'my_role';
```

Keywords

```
CREATE, USER
```

Best Practices

1. Users and roles

   The role specified for creating a user must already exist, which can be used to create a role by CREATE ROLE command.

DROP-ROLE

DROP ROLE Description

This statement is used to drop a role.

```
DROP ROLE role1;
```

Drop a role will not affect the grants of users who was previously of the role. It is only equivalent to decoupling the role from the user. The grants that the user has obtained from this role will not change.

Example

1. Drop a role

```
DROP ROLE write_role;
```

Keywords

```
DROP, ROLE
```

CREATE-ROLE

CREATE ROLE

Description

This statement is used to create a role.

```
CREATE ROLE role1;
```

This statement is used to create a role with no grant, grants can be done by `GRANT` command later.

Example

1. Create a role

```
CREATE ROLE read_role;
```

Keywords

```
CREATE, ROLE
```

GRANT

GRANT Description

The GRANT command is used to grant specified privilege to the specified user or role.

Grant privilege for databases and tables:

```
GRANT privilege_list
ON db_name[.tbl_name]
TO user_identity [ROLE role_name]
```

Grant resource privilege:

```
GRANT privilege_list
ON RESOURCE resource_name
TO user_identity [ROLE role_name]
```

- `privilege_list`

  List of privileges to be granted, separated by comma.

  Currently PALO supports the following privileges:

  - `ADMIN_PRIV` : All privileges except node management.
  - `GRANT_PRIV` : Privilege to grant privileges, including creating and dropping users and roles, granting and revoking privileges, setting passwords, etc.
  - `SELECT_PRIV` : Privilege to read specified library or table
  - `LOAD_PRIV` : Privilege to load specified library or table
  - `ALTER_PRIV` : Privilege to change the schema of specified library or table
  - `CREATE_PRIV` : Privilege to create specified library or table
  - `DROP_PRIV` : Privilege to drop specified library or table
  - `USAGE_PRIV` : Privilege to use specified resource

  In addition, there are two syntax sugar privileges that can be used for quick granting:

  - `ALL`

Similar to read and write privileges, which is equivalent to granting:

`SELECT_PRIV,LOAD_PRIV,ALTER_PRIV,CREATE_PRIV,DROP_PRIV`

these privileges.

- `READ_ONLY`

  Similar to read-only privilege, which is equivalent to `SELECT_PRIV`。

Classification of privileges:

1. Library and table privileges: SELECT_PRIV , LOAD_PRIV , ALTER_PRIV , CREATE_PRIV , DROP_PRIV
2. Resource privilege: USAGE_PRIV

- `db_name[.tbl_name]` supports the following three forms:
  1. `*.*` : Privileges can be applied to all libraries and all tables in library.
  2. `db.*` : Privileges can be applied to all tables under specified library.
  3. `db.tbl` : Privileges can be applied to specified table under specified library.

> The specified library or table here can be a nonexistent library or table.

- `resource_name` supports the following two forms:
  1. `*` : Privileges are applied to all resources.
  2. `resource` : Privileges are applied to a specified resource.

  The specified resource here can be a nonexistent resource.

- `user_identity`

  The `user_identity` syntax here is the same as the syntax in `CREATE USER` and must be the `user_identity` created by using CREATE USER. The `host` in`user_identity` can be a domain name. If it is a domain name, the effective time of privilege may be delayed for about 1min.

  The privilege can also be granted to specified `ROLE`, if the specified `ROLE` does not exist, then it will be automatically created.

Example

1. Grant privileges on all libraries and tables to users

```
GRANT SELECT_PRIV ON *.* TO 'jack'@'%';
```

2. Grant privileges on specified libraries and tables to users

```
GRANT SELECT_PRIV,ALTER_PRIV,LOAD_PRIV ON db1.tbl1 TO 'jack'@'192.8.%';
```

3. Grant privileges on specified libraries and tables to users

```
GRANT LOAD_PRIV ON db1.* TO ROLE 'my_role';
```

4. Grant privileges on all resources to users

```
GRANT USAGE_PRIV ON RESOURCE * TO 'jack'@'%';
```

5. Grant privileges on specified resources to users

```
GRANT USAGE_PRIV ON RESOURCE 'odbc_resource' TO 'jack'@'%';
```

6. Grants privileges on use of specified resources to roles

```
GRANT USAGE_PRIV ON RESOURCE 'odbc_resource' TO ROLE 'my_role';
```

Keywords

```
GRANT
```

REVOKE

REVOKE Description

Used to revoke specified privileges of specified user or role.

Revoke the privileges on libraries and tables:

```
REVOKE privilege_list
ON db_name[.tbl_name]
FROM user_identity [ROLE role_name]
```

Revoke the privileges on resources:

```
REVOKE privilege_list
ON RESOURCE resource_name
FROM user_identity [ROLE role_name]
```

- `user_identity`

  The `user_identity` syntax here is the same as the syntax in `CREATE USER` and must be the `user_identity` created by using `CREATE USER` . The `host` in`user_identity` can be a domain name. If it is a domain name, the revoke time of privilege may be delayed for about 1min.

The privilege of specified ROLE can also be revoked, and the ROLE executed must exist.

Example

1. Revoke the privilege of user jack on database testDb

```
REVOKE SELECT_PRIV ON db1.* FROM 'jack'@'192.%';
```

2. Revoke the privilege of user jack on resource odbc_resource

```
REVOKE USAGE_PRIV ON RESOURCE 'odbc_resource' FROM 'jack'@'192.%';
```

Keywords

```
REVOKE
```

Auxiliary Commands

RESUME-ROUTINE-LOAD

RESUME ROUTINE LOAD

Description

Used to resume a paused job named as Routine Load. The resumed job will be consumed from the offset that has been consumed already.

```
RESUME ROUTINE LOAD FOR job_name
```

Example

1. Pause a routine load job named as test1.

```
RESUME ROUTINE LOAD FOR test1;
```

Keywords

```
RESUME, ROUTINE, LOAD
```

CANCEL-LOAD

CANCEL LOAD

Description

This statement is used to cancel the load job of the specified label or cancel the load jobs in batch through fuzzy matching.

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

Example

1. Cancel the load job labelled as `example_db_test_load_label` on the database example_db

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

2. Cancel the load jobs containing example_ on database example_db.

```
CANCEL LOAD
FROM example_db
WHERE LABEL like "example_";
```

Keywords

```
CANCEL, LOAD
```

Best Practices

1. Only load jobs in PENDING, ETL, LOADING status that have not been completed yet can be cancelled.
2. When executing cancel in batch, Palo will not guarantee that all corresponding load jobs can be cancelled. That is, it is possible that only some load jobs can be cancelled successfully. Users can view the job status by using SHOW LOAD statement and try to repeat the execution of CANCEL LOAD statement.

ALTER-ROUTINE-LOAD

ALTER ROUTINE LOAD Description

This syntax is used to modify the routine load jobs that have been created.

Only jobs in PAUSED state can be modified.

```
ALTER ROUTINE LOAD FOR [db.]job_name
[job_properties]
FROM data_source
[data_source_properties]
```

- `[db.]job_name`

  Specify the job name to be modified.

- `tbl_name`

  Specify the name of the table to be loaded.

- `job_properties`

  Specify the job parameters to be modified. Currently only the following parameters can be modified:

  1. `desired_concurrent_number`

2. max_error_number
3. max_batch_interval
4. max_batch_rows
5. max_batch_size
6. jsonpaths
7. json_root
8. strip_outer_array
9. strict_mode
10. timezone

- data_source

  Data source type. Currently support:

  KAFKA

- data_source_properties

  Related properties of the data source. Currently only support:

  1. kafka_partitions
  2. kafka_offsets
  3. Customize property, such as property.group.id

> Note:
>
> 1. kafka_partitions and kafka_offsets are used to modify the offset of kafka partition to be consumed, only the partition that has already consumed can be modified. It is allowed to add new partition.

Example

1. Modify desired_concurrent_number as 1

```
ALTER ROUTINE LOAD FOR db1.label1
PROPERTIES
(
   "desired_concurrent_number" = "1"
);
```

2. Modify desired_concurrent_number as 10, modify the offset of partition, modify group id.

```
ALTER ROUTINE LOAD FOR db1.label1
PROPERTIES
(
    "desired_concurrent_number" = "10"
)
FROM kafka
(
    "kafka_partitions" = "0, 1, 2",
    "kafka_offsets" = "100, 200, 100",
    "property.group.id" = "new_group"
);
```

Keywords

```
ALTER, ROUTINE, LOAD
```

Best Practices

This command can be used to correct the offset of consumption, or modify the mapping, transformation and filtering rules of columns after the Kafka data format is changed.

STOP-ROUTINE-LOAD

STOP ROUTINE LOAD

Description

User stops a job named as Routine Load. The job stopped cannot re-run.

```
STOP ROUTINE LOAD FOR job_name;
```

Example

1. Stop a routine load job named as test1.

```
STOP ROUTINE LOAD FOR test1;
```

Keywords

```
STOP, ROUTINE, LOAD
```

PAUSE-ROUTINE-LOAD

PAUSE ROUTINE LOAD

Description

Used to pause Routine Load job. The jobs paused can be re-run by RESUME command.

```
PAUSE ROUTINE LOAD FOR job_name
```

Example

1. Pause the routine load job for test1.

```
PAUSE ROUTINE LOAD FOR test1;
```

Keywords

```
PAUSE, ROUTINE, LOAD
```

Alias

When you write the name of lists, columns, or expressions that contain columns in a query, you can assign them an alias at

the same time. Whenv using the list 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 list, column or expression in the select list or the from list. The AS keyword is optional. The user can specify an 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 case sensitive.

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;
```

Comments

Doris supports the SQL comments.

- Single-line comments: Statements beginning with may be recognized as comments and ignored. Single-line coments can be made independently or appear after partial or complete statements of other statements.
- Multi-line comments: The text between '/*' and '*/' is recognized as comments and ignored. A multiline comment can occupy a single line or multiple lines, or appear in the middle, front, or end part 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;
```

Data Type

TINYINT Data Type

Length: Signed integer with a length of 1 byte.

Range: [128, 127]

Conversion: Doris 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 a length of 2 bytes.

Range: [-32768, 32767]

Conversion: Doris 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 a length of 4 bytes.

Range: [-2147483648, 2147483647]

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

For 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 a length of 8 bytes.

Range: [-9223372036854775808, 9223372036854775807]

Conversion: Doris 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 a length of 16 bytes.

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

Conversion: Doris can automatically convert this type to a floatingpoint type. Use the CAST () function to convert it to TINYINT, SMALLINT, INT, BIGINT, CHAR

Example:

```
 mysql> select cast(92233720368547758234234 2 as double);
+-----------------------------------------+
| CAST(9223372036854775823423 42 AS DOUBLE) |
+-----------------------------------------+
|                  9.223372036854776e23 |
+-----------------------------------------+
1 row in set (0.05 sec)
```

FLOAT Data Type

Length: Floating point type with a length of 4 bytes.

Range: 3.40E+38 ~ +3.40E+38。

Conversion: Doris can automatically convert 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 length of 8 bytes.

Range: 1.79E+308 ~ +1.79E+308。

Conversion: Doris 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 the 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], and 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. Display the value range of the specified decimal.

   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: ['10000101', '99991231']. The default print format is 'YYYY-MM-DD'.

DATETIME Data Type

Range: ['10000101 00:00:00', '99991231 00:00:00']. The default print format is 'YYYY-MM-DD HH:MM:SS'.

CHAR Data Type

Range: Char[(length)], a fixed-length string, whose length ranges from 1 to 255 and is 1 by default.

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), a variable length string with a length range of 1~65535 characters.

Conversion: Users can convert CHAR type to TINYINT, SMALLINT, INT, BIGINT, LARGEINT, DOUBLE, DATE or DATETIME type through the 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) |
+-------------------------------+
| 1/1/2011 12:00:00 AM          |
+-------------------------------+
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

The HLL (HyperLogLog) type is a binary type. The HLL type can only be used for aggregation table, and the aggregation type must be specified as HLL_UNION.

The HLL type is mainly used for pre-aggregation of data in inaccurate and fast deduplication scenarios.

The HLL column can only be queried or used through the matching hll_union_agg, hll_cardinality, and hll_hash.

BITMAP Data Type

The BITMAP type is a binary type. The BITMAP type can only be used for aggregation table, and the aggregation type must be specified as BITMAP_UNION.

The BITMAP type is mainly used for pre-aggregation of data in precise deduplication scenarios. Also, it can also be used to store user IDs in user portrait scenes.

The BITMAP column can only be queried and used through the supporting BITMAP feature.

Literal Constant

Each data type in Doris corresponds to a Literal of such type. The user can specify Literal in SQL statements, such as in the selected list, WHERE clauses, and function's parameters.

Literal Constant of Numbers

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

The literal constant of floatingpoint type, such as DOUBLE, is a series of numbers, optionally with a decimal dot, i.e., . character.

If 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.

Literal Constant of Strings

String literal constants are put in a 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 character of ASCII code, which is different from the NULL in SQL.
- \Z means the endoftext character in dos environment.
- \% and _ are used to escape wildcard characters in strings passed to the operator LIKE
- \\ prevents the backslash symbol from being interpreted as an escape character.
- 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 characters listed above, the character should remain unchanged and should not be escaped.

Literal Constant of Date

Doris automatically converts CHAR type literal constants to DATE type literal constants. The time type literal constant accepted by Doris is in the input format of YYYYMMDD 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 Operators

SQL operator is 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.
- %: modulus 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 returns the bitwise-AND, bitwise-OR, and bitwise-XOR operation results for two operands. Both operands are required to be an integral type. If the types of the two operands of the bitwise operator are different, the smaller operand is increased to the bigger operand, and then the corresponding bitwise operation is performed. Multiple arithmetic operators may appear in one expression. The user can use parentheses to enclose the corresponding arithmetic expression. Typically, the arithmetic operator does not have corresponding mathematical functions to express the same feature as the arithmetic operator. For example, we don't use the MOD() function to express the feature of the % operator. On the contrary, the mathematical function does not have a corresponding arithmetic operator. For example, the power function POW() does not have a corresponding ** exponentiation operator. Users can learn about which arithmetic functions we support through the chapter on mathematical functions.

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 you use string operands. Long strings starting with the upper bound do 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 ().

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, which is the same as !=. Operators IN and BETWEEN provide a shorter expression to describe the comparison of such relationships as equal, less than, size, etc.

In Operator

The In operator 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 Doris to optimize.

For example:

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

Like Operator

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 operators return a BOOL value. The logical operators include 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 is a binary operator, which 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. You need to use regular expressions meeting POSIX standard. ^ is used 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, and ? to express fractal greedy expressions, etc. Regular expressions need to match a complete 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 $ can be omitted. Operators RLKIE and REGEXP are synonymous. The | operator is an optional operator. The regular expressions on both sides of the | operator only need to satisfy the conditions on one side. The | operator and the regular expressions on both sides usually need to be enclosed in ().

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)
```

SQL-Manual

Doris provides online and offline SQL manuals.

You can view the online SQL manual using the Help command after connecting to Doris. 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 which details the syntax of SQL.

Service Level AgreementSLA

Palo Service Level Agreement SLA（V1.0）

Effective date of the agreement: September 1, 2020

The Service Level Agreement (hereinafter referred to as "SLA") stipulates the service availability level index and compensation schemes of the Palo data repository provided by Baidu intelligent cloud to users.

1. Definition

Service cycle: a service cycle is a natural month.

Total minutes of service cycle: calculated according to the total number of days *24 (hours)* 60 (minutes) in the service cycle.

Unavailable minutes of service: When all continuous attempts (such as data writing and query) to establish a connection with the specified Palo data repository cluster fail, and the status lasts for more than 5 minutes, it is considered that the Palo cluster service is not available within the minutes. In a service cycle, the sum of the unavailable minutes of Palo cluster service (different operations in the same minute are not superimposed) is the unavailable minutes of service.

Monthly service charge: It is the total service charge paid by a user for a single Palo cluster in a natural month. If the user pays the service charge for multiple months at one time, the monthly service charge will be calculated by apportionment

according to the number of months purchased.

2. Service availability

2.1 Calculation formula for service availability

Service availability = (1- unavailable minutes of service/ total minutes of service cycle) * 100%

2.2 Service availability commitments

The availability of Palo data repository service shall not be less than 99.95%, if the Palo service fails to meet the above availability commitments, the user can obtain compensation according to Article 3 of this agreement.

3. Service compensation clauses

3.1 Compensation standards

According to the monthly service availability of the Baidu Palo data repository of a certain area according a certain Baidu Intelligent Cloud account, the amount of compensation is calculated according to the standards in the table below. The compensation method is limited to the voucher for purchasing Baidu Palo data repository, and the total amount of compensation shall not exceed 25% of the monthly service charge paid by the user in the region under the account in the month of failure to meet the service availability commitment.

Service availability	Amount of compensation voucher
Less than 99.95% but equal to or higher than 99.00%	10% of monthly service charge
Less than 99.00% but equal to or higher than 95.00%	25% of monthly service charge
Less than 95.00%	100% of monthly service charge

3.2 Time limit for application for compensation

The customer may apply for compensation after the fifth (5th) working day of each month for services that did not reach availability in last month. Application for compensation must be limited to two (2) months after the end of the relevant month in which the message service is not available. Application for compensation beyond the time limit will not be accepted.

3.3 Exceptions

The scope of compensation does not include the unavailability caused by the following reasons:

(1) Baidu Intelligent Cloud notifies users in advance of system maintenance, including cutting, maintenance, upgrading and simulation fault drill.

(2) Any network or equipment failure or configuration adjustment caused by any equipment not belonging to Baidu intelligent cloud.

(3) The user's application or data information is attacked by hackers;

(4) Data, passwords and other information are lost or leaked due to improper maintenance or confidentiality of users;

(5) The negligence of the user or the operation authorized by the user;

(6) Users do not follow Baidu Intelligent Cloud product documentation or use suggestions, such situations include abnormal operation state (i.e., non-green state) of the basic state of the customer cluster, high utilization rate of CPU, high memory load, and excessive disk usage (above values are recommended to remain below 50%).

(7)Unavailability of when the data are erased due to the occurrence of downtime data, and the unavailability caused by the startup dependency of local disk and the data;

(8) Service unavailability caused by human configuration operation of Palo data repository;

(9) Service unavailability caused by instance change and restart, forced restart of Palo data repository cluster;

(10) Abnormal use of Baidu Palo data repository caused by force majeure and accidents or other non Baidu Intelligent Cloud causes.

4. Other instructions

(1) Baidu reserves the right of final interpretation of this agreement to the extent permitted by laws and regulations.

(2) As soon as this agreement is released, it will take effect immediately, Baidu intelligent cloud has the right to modify the terms of the SLA. If there is any modification to this SLA clause, Baidu intelligent cloud will notify you by way of website publicity or mail. If you disagree with the modification made by Baidu intelligent cloud to SLA, you have the right to stop using the Palo data repository service, if you continue to use the Palo data repository service, then it is deemed that you accept the modified SLA.

(3) Under this agreement, all notices of Baidu intelligent cloud can be carried out through web publicity, station letter, e-mail, mobile phone text or other forms. These notices shall be deemed to have been delivered to the addressee on the date of delivery. Baidu intelligent cloud is not responsible for any loss of service modification or termination clause of Baidu Intelligent Cloud due to users' failure to learn in time.

(4) The conclusion, execution and interpretation of this Agreement and the settlement of disputes shall be governed by Chinese laws and shall be subject to the jurisdiction of Chinese courts. If any dispute arises between the parties concerning the contents of this agreement or its implementation, both parties shall try to settle the dispute through friendly negotiation as far as possible; If the negotiation fails, either party may bring a lawsuit to the People's Court of Haidian District, Beijing.

(5) This Agreement constitutes the entire agreement of both parties on the agreed matters and other related matters of this agreement. Except for the provisions of this agreement, no other rights are granted to each party of this agreement.

(6) If any agreement in this Agreement is invalid or unenforceable in whole or in part for any reason, the rest of this Agreement shall remain valid and binding.

(7) Refer to "Rights and obligations of users" in *User service agreement* for the user binding terms.

(8) Refer to the relevant clauses of "Exemption statements" in *User service agreement* for details of the service provider's exemption clauses.

Open Source Zone

Open Source Version

As the main maintenance team of Apache Doris community, Baidu Palo team also maintains **3-bit iterative Version (tags)** (What is 3-bit version) based on the official release version of Apache Doris, including **fast Bug repair and new function update**.

These 3-bit iterations have been tested in Baidu internally and have been made online, which are recommendable.

Open-source users can download our 3-bit iterative version for free. Please refer to the official Apache Doris website at the end of the document for the installation and deployment method.

Download open-source version

- In some cases, the user may not be able to get the binary file of Doris smoothly through source code compilation. Here we provide the pre-compiled binary download of the corresponding 3-bit version.
- We **strongly recommend** users to generate binary files through source code compilation.
- The pre-compiled binary files provided here only run on CentOS 7.3, Intel (R) Xeon (R) Gold 6148 CPU @ 2.40GHz for

execution. In other systems or CPU models, the files may not run because of different glibc versions or different instruction sets supported by CPU.

- The FE part of the pre-compiled binary file is compiled with Oracle JDK 1.8, so please make sure that Oracle JDK 1.8 is still used at runtime.
- Pre-compile includes the following components
    1. Frontend
    2. Backend
    3. Broker
    4. Frontend plugins jars
    5. Spark-Doris-Connector jars

Version numbers	Size	Download link	Update time
0.12.21	447MB	Link	2020-08-11
0.13.9	547MB	Link	2020-10-21
0.13.11	552MB	Link	2020-11-15
0.13.15	554MB	Link	2021-01-05

Change Log

- 0.13.15
    1. New functions:
        - A new Join Reorder algorithm is added, which has obvious optimization effect in TPCDS 17, 25, 37, 54, 82, 84, 85 and other complex Join queries. Join Reorder.
        - It supports the calculation of all constant expressions through BE to solve the problem that FE is not able to calculate constant expressions. Constant expression folding.
        - It supports Export the data to ODBC external table.
        - TopN aggregate function is added.
    2. Optimization:
        - It can optimize the data reading logic of storage engine to improve the query performance, including version merging, predicate push down and so on.
        - Through parameter `fuzzy_ Parse` , it can optimize part loading speed of scenario JSON Data Loading.
    3. Serious bug fix:
        - Fix the problem that loading JSON format data may cause BE memory leak.
        - Fix the problem that BE execution of compact logic may cause disk space not to be released.
        - Fix the problem of BE downtime caused by Delete condition containing predicate`IS NULL/IS NOT NULL`.
        - Fix the problem that Routine Load consumes Kafka data and may lose data in some cases.
        - Fix some query problems.

Instructions

All 3-bit iterative version of the source code can be seen here Baidu-Doris

Currently, the official release version of Apache Doris is:

- 0.9.0
- 0.10.0
- 0.11.0
- 0.12.0
- 0.13.0

This repository mainly releases 3-bit iterative version based on 2-bit official version. For example:

- DORIS-0.9.22-release
- DORIS-0.10.23-release
- DORIS-0.11.44-release
- DORIS-0.12.21-release
- DORIS-0.13.11-release

All 3-bit versions can be safely upgraded from the corresponding official 2-bit version. The 3-bit version itself is compatible and can be upgraded safely. Here are the examples:

- Official version `0.12.0-rc02` can be upgraded to `DORIS-0.12.21-release`
- `DORIS-0.11.10-release` can be upgraded to `DORIS-0.11.44-release`
- `DORIS-0.11.44-release` can be upgraded to `DORIS-0.12.21-release`

It is recommended to upgrade to the latest 3-bit version before upgrading to the 2-bit version. Here are the examples:

1. Current version is `DORIS-0.11.10-release`, if you want to upgrade it to version 0.12 .
2. First upgrade to `DORIS-0.11.44-release`, that is, the latest 3-bit version of 0.11
3. Then upgrade to `DORIS-0.12.21-release`, that is, the latest 3-bit version of 0.12.

The 3-bit iterative version can also be safely upgraded with the official 2-bit version. For example, the following upgrade sequence is safe:

Upgrade sequence	Version	Instruction
1	DORIS-0.11.10-release	Baidu library
2	DORIS-0.11.44-release	Baidu library
3	0.12-rc02	Official library
4	DORIS-0.12.21-release	Baidu library

Docker compile environment image download

We recommend that open-source users compile Doris by themselves through the source code.

But there are some reasons that cause slow download of `imagedocker.io/apachedoris/doris-dev:build-env-1.2` through the method of `docker pull`. We can download the image locally through the following links, and then load the image through `docker load` command:

Download docker.io/apachedoris/doris-dev:build-env-1.2

`docker load --input apachedoris-build-env-1.2`

After this, we can view the image through `docker images`.

Open-source link

Apache Doris official website: http://doris.incubator.apache.org