Clickhouse Materialized view

Based on clickhouse 20.3.5.21 version

1. MergeTree and SummingMergeTree

Create source table

a2-test-clickhouse-1-223.sh :) show create table liuyang_ck_test1\G

statement: *CREATE TABLE default.liuyang_ck_test1 (`ts` DateTime, `userid` UInt32, `bytes` UInt32) ENGINE = MergeTree PARTITION BY toYYYYMM(ts) ORDER BY (userid, ts) SETTINGS index_granularity = 8192*

create two different engine tables (MergeTree / SummingMergeTree)

a2-test-clickhouse-1-223.sh :) show create table liuyang_ck_view_test1\G

***statement**: CREATE MATERIALIZED VIEW default.liuyang_ck_view_test1 (`day` DateTime('Asia/Shanghai'), `userid` UInt32, `downloads` UInt64, `bytes` UInt64) ENGINE = SummingMergeTree PARTITION BY toYYYYMM(day) ORDER BY (userid, day) SETTINGS index_granularity = 8192 AS SELECT toStartOfDay(ts) AS day, userid, count(*) AS downloads, sum(bytes) AS bytes FROM default.liuyang_ck_test1 GROUP BY userid, day*

a2-test-clickhouse-1-223.sh :) show create table liuyang_ck_view_test2\G

***statement:** CREATE MATERIALIZED VIEW default.liuyang_ck_view_test2 (`day` DateTime('Asia/Shanghai'), `userid` UInt32, `downloads` UInt64, `bytes` UInt64) ENGINE = MergeTree PARTITION BY toYYYYMM(day) ORDER BY (userid, day) SETTINGS index_granularity = 8192 AS SELECT toStartOfDay(ts) AS day, userid, count(*) AS downloads, sum(bytes) AS bytes FROM default.liuyang_ck_test1 GROUP BY userid, day*

2. Insert single rows and check the different between two tables with different engine

a2-test-clickhouse-1-223.sh :) insert into liuyang_ck_test1 values ('2020-01-01 00:00:00',10000,2000);
####check the source table
A2-test-clickhouse-1-223.sh :) select * from liuyang_ck_test1 \G

Row 1:
——————
ts: 2020-01-01 00:00:00
userid: 10000
bytes: 2000

a2-test-clickhouse-1-223.sh :) select * from liuyang_ck_view_test1 \G

Row 1:
——————
day: 2020-01-01 00:00:00
userid: 10000
downloads: 1
bytes: 2000

a2-test-clickhouse-1-223.sh :) select * from liuyang_ck_view_test2 \G

Row 1:

——————————

day: 2020-01-01 00:00:00
userid: 10000
downloads: 1
bytes: 2000

Insert the same date-type rows again

####There's no difference

a2-test-clickhouse-1-223.sh :) select * from liuyang_ck_view_test1 \G

Row 1:

——————————

day: 2020-01-01 00:00:00
userid: 10000
downloads: 1
bytes: 2000

Row 2:

——————————

day: 2020-01-01 00:00:00
userid:10000

downloads: 1
bytes: 2000
a2-test-clickhouse-1-223.sh :) select * from liuyang_ck_view_test2 \G

Row 1:
——————
day: 2020-01-01 00:00:00
userid: 10000
downloads: 1
bytes: 2000

Row 2:
——————
day: 2020-01-01 00:00:00
userid: 10000
downloads: 1
bytes: 2000

OPTIMZE TABLE find that SummingMergeTree table merge the rows

a2-test-clickhouse-1-223.sh :) optimize table liuyang_ck_view_test1
a2-test-clickhouse-1-223.sh :) optimize table liuyang_ck_view_test2
a2-test-clickhouse-1-223.sh :) select * from liuyang_ck_view_test1 \G

Row 1:

———————

day: 2020-01-01 00:00:00

userid: 10000

downloads: 2

bytes: 4000

a2-test-clickhouse-1-223.sh :) select * from liuyang_ck_view_test2 \G

Row 1:

———————

day: 2020-01-01 00:00:00

userid: 10000

downloads: 1

bytes: 2000

Row 2:

———————

day: 2020-01-01 00:00:00

userid: 10000

downloads: 1

bytes: 2000

3. Insert multi-rows one time and check the difference between the two tables

Source table

```
a2-test-clickhouse-1-223.sh :) select * from liuyang_ck_view_test1 \G

Row 1:
──────
day: 2020-01-01 00:00:00
userid: 10000
downloads: 2
bytes: 4000
```

```
a2-test-clickhouse-1-223.sh :) select * from liuyang_ck_view_test2 \G

Row 1:
──────
day: 2020-01-01 00:00:00
userid: 10000
downloads: 1
bytes: 2000
```

Row 2:

———————

day: 2020-01-01 00:00:00

userid: 10000

downloads: 1

bytes: 2000

Insert three rows one time and check the merge action

a2-test-clickhouse-1-223.sh :) insert into liuyang_ck_test1 values ('2020-01-01 00:00:00',10000,2000),('2020-01-01 00:00:00',10000,2000),('2020-01-01 00:00:00',10000,2000)

a2-test-clickhouse-1-223.sh :) select * from liuyang_ck_view_test1 \G

Row 1:

———————

day: 2020-01-01 00:00:00

userid: 10000

downloads: 2

bytes: 4000

Row 2:

——————————

day: 2020-01-01 00:00:00
userid: 10000
downloads: 3
bytes: 6000

a2-test-clickhouse-1-223.sh :) select * from liuyang_ck_view_test2 \G

Row 1:

——————————

day: 2020-01-01 00:00:00
userid: 10000
downloads: 1
bytes: 2000

Row 2:

——————————

day: 2020-01-01 00:00:00
userid: 10000
downloads: 1
bytes: 2000

Row 3:

———————

day: 2020-01-01 00:00:00

userid: 10000

downloads: 3

bytes: 6000

Find that two tables are both merging downloads/bytes rows at one transaction , three rows which were inserted by this transaction are met "count(*) as downloads,sum(bytes) as bytes"

####OPTIMEZ TABLE again

a2-test-clickhouse-1-223.sh :) optimize table liuyang_ck_view_test1

a2-test-clickhouse-1-223.sh :) optimize table liuyang_ck_view_test2

a2-test-clickhouse-1-223.sh :) select * from liuyang_ck_view_test1 \G

Row 1:

———————

day: 2020-01-01 00:00:00

userid: 10000

downloads: 5

bytes: 10000

a2-test-clickhouse-1-223.sh :) select * from liuyang_ck_view_test2 \G

Row 1:

——————————

day: 2020-01-01 00:00:00
userid: 10000
downloads: 1
bytes: 2000

Row 2:

——————————

day: 2020-01-01 00:00:00
userid: 10000
downloads: 1
bytes: 2000

Row 3:

——————————

day: 2020-01-01 00:00:00
userid: 10000
downloads: 3
bytes: 6000

The result is as expected, SummingMergeTree merges downloads/bytes column data again

Summary:

- Using SummingMergeTree can meet the general statistical requirements in most cases, that is, the data of the materialized view will reach the final consistency. In the process of final merging data blocks of Clickhouse, there is a certain "difference" in the logical query of the materialized view and the actual table data. , But after using OPTIMZER TABLE to force the merge, the query display consistent with the main table is achieved (this itself is actually similar to the materialized view refresh method of databases such as Oracle).

```
a2-test-clickhouse-1-223.sh :) select    * from    liuyang_ck_view_test1 \G

Row 1:
──────────
day: 2020-01-01 00:00:00
userid: 10000
downloads: 2
bytes: 4000

Row 2:
──────────
day: 2020-01-01 00:00:00
userid: 10000
downloads: 3
bytes: 6000

a2-test-clickhouse-1-223.sh :)    select    toStartOfDay(ts) AS day,userid,count(*) as downloads,sum(bytes) as bytes    from    liuyang_ck_test1    group by userid,day\G
```

Row 1:

——————————

day: 2020-01-01 00:00:00

userid: 10000

downloads: 5

bytes: 10000

The above query found that even if the same logical SQL query is used at the same point in time, the materialized view and the source table data are different, but this is in line with expectations

- Based on this kind of performance, we found that the implementation of the underlying materialized view of Clickhouse is relatively rudimentary. It only supports the current LOG merge, and does not support the transaction-based global real-time data refresh. The subsequent overall data merge also depends on the underlying data The merging of blocks (of course, you can use optimze table to force trigger), which increases the difficulty of choosing the view engine itself. Compared with the materialized views of mature relational database systems such as Oracle (supporting full and incremental based on LOG/on commit and demand refreshing methods), CK's materialized views support fewer functions and require more work on the business side to optimize data display

- Check the difference between Oracle and Clickhouse (Simple is based on Oracle's fast refresh on commit materialized)

Clickhouse：

```
a2-test-clickhouse-1-223.sh :) select  *  from  liuyang_ck_view_test1 – clickhouse MV  \G
```

Row 1:

——————————

day: 2020-01-01 00:00:00
userid: 10000
downloads: 5
bytes: 10000

a2-test-clickhouse-1-223.sh :) insert into liuyang_ck_test1 values ('2020-01-01 00:00:00',10000,2000);
a2-test-clickhouse-1-223.sh :) select * from liuyang_ck_view_test1 – clickhouse MV \G

Row 1:

——————————

day: 2020-01-01 00:00:00
userid: 10000
downloads: 5
bytes: 10000

Row 2:

——————————

day: 2020-01-01 00:00:00
userid: 10000
downloads: 1
bytes: 2000

Optimize table to the final consistency

```
a2-test-clickhouse-1-223.sh :) optimize table    liuyang_ck_view_test1;
a2-test-clickhouse-1-223.sh :) select    *    from liuyang_ck_view_test1 – clickhouse MV    \G


Row 1:
——————
day: 2020-01-01 00:00:00
userid: 10000
downloads: 6
bytes: 12000
```

Oracle: choose the materialized view base on rowed and fast refresh on commit

```
sys@SZTEST01>create table H2.liuyang_ck_test1 (ts date ,userid int,bytes int);
sys@SZTEST01>insert into H2.liuyang_ck_test1 values (to_date('2020-01-01 00:00:00','yyyy-mm-dd hh24:mi:ss'),10000,2000);
sys@SZTEST01>insert into H2.liuyang_ck_test1 values (to_date('2020-01-01 00:00:00','yyyy-mm-dd hh24:mi:ss'),10000,2000);
sys@SZTEST01>insert into H2.liuyang_ck_test1 values (to_date('2020-01-01 00:00:00','yyyy-mm-dd hh24:mi:ss'),10000,2000);
sys@SZTEST01>commit;
sys@SZTEST01>select    ts ,userid,count(*) as downloads,sum(bytes) as bytes from H2.liuyang_ck_test1    group by userid,ts
```

```
TS                   USERID    DOWNLOADS         BYTES
------------------- ---------- ---------- ----------
01-JAN-20             10000              3          6000
```

```
sys@SZTEST01>create materialized view log on H2.liuyang_ck_test1 with rowid,SEQUENCE(ts,userid,bytes) including new values ;
sys@SZTEST01>create materialized view H2.liuyang_ck_view_test1 build immediate refresh FAST on commit with rowid as (select    ts ,userid,count(*)
as downloads,sum(bytes) as bytes from H2.liuyang_ck_test1    group by userid,ts);
sys@SZTEST01>select * from H2.liuyang_ck_view_test1;
```

```
TS                   USERID    DOWNLOADS         BYTES
------------------- ---------- ---------- ----------
01-JAN-20             10000              3          6000
```

```
sys@SZTEST01>insert into H2.liuyang_ck_test1 values (to_date('2020-01-01 00:00:00','yyyy-mm-dd hh24:mi:ss'),10000,2000);
sys@SZTEST01>commit;
```

```
sys@SZTEST01>select * from H2.liuyang_ck_view_test1;
```

```
TS                   USERID    DOWNLOADS         BYTES
------------------- ---------- ---------- ----------
01-JAN-20             10000              4          8000
```

It can be seen that Oracle's approach based on materialized views is very flexible. Although this commit-based refresh consumes resources, for special low-frequency update services, higher consistency brings a different experience to business queries.

4. Advanced usage of materialized views - AggregatingMergeTree

This kind of aggregation engine is very suitable for CK's materialized view scene. It tries to construct a "data cube", which greatly simplifies the complexity of the query. Here is an example of pv/uv mixed calculation

```
a2-test-clickhouse-1-223.sh :)create table ck_pv_uv (name String,page String,page_location UInt32 ,clicks UInt32, gmt date) engine=MergeTree order by gmt
insert into    ck_pv_uv values    ('甲','a',1,1,'2020-04-22');
insert into    ck_pv_uv values    ('甲','a',2,1,'2020-04-22');
insert into    ck_pv_uv values    ('甲','b',1,1,'2020-04-22');
insert into    ck_pv_uv values    ('乙','a',1,1,'2020-04-22');
insert into    ck_pv_uv values    ('乙','b',1,1,'2020-04-22');
insert into    ck_pv_uv values    ('乙','a',3,1,'2020-04-22');
insert into    ck_pv_uv values    ('乙','c',2,1,'2020-04-22');
insert into    ck_pv_uv values    ('丙','a',1,1,'2020-04-22');
insert into    ck_pv_uv values    ('丙','a',2,1,'2020-04-22');
insert into    ck_pv_uv values    ('丙','b',1,1,'2020-04-22');
insert into    ck_pv_uv values    ('丙','b',3,1,'2020-04-22');
insert into    ck_pv_uv values    ('丁','a',1,1,'2020-04-22');
insert into    ck_pv_uv values    ('丁','a',3,1,'2020-04-22');
```

Pv based on page , page_location

```
a2-test-clickhouse-1-223.sh :) select page,page_location,count(clicks) from ck_pv_uv group by    page,page_location\G
```

Row 1:
———————
page: b
page_location: 1
count(clicks): 3

Row 2:
———————
page: c
page_location: 2
count(clicks): 1

Row 3:
———————
page: a
page_location: 3
count(clicks): 2

Row 4:
———————
page: b
page_location: 3
count(clicks): 1

Row 5:
———————

page: a
page_location: 1
count(clicks): 4

Row 6:
———————
page: a
page_location: 2
count(clicks): 2

Uv based on name ,page

a2-test-clickhouse-1-223.sh :) select page ,uniq(name) from ck_pv_uv group by page\G

Row 1:
———————
page: b
uniq(name): 3

Row 2:
———————
page: c
uniq(name): 1

Row 3:

page: a
uniq(name): 4

Create AggregatingMergeTree engine Materialized view

a2-test-clickhouse-1-223.sh :) create materialized view ck_pv_uv_view Engine=AggregatingMergeTree order by gmt POPULATE as select page,page_location,gmt,uniqState(name) as uv ,sumState(clicks) as pv from ck_pv_uv group by page,page_location ,gmt
a2-test-clickhouse-1-223.sh :) select * from ck_pv_uv_view\G

Row 1:

page: a
page_location: 1
gmt: 2020-04-22
uv:防 µXOAv□Ọ
pv:

Row 2:

page: b
page_location: 1
gmt: 2020-04-22
uv:防 µX2§Ọ
pv:

Row 3:

———————

page: c
page_location: 2
gmt: 2020-04-22
uv: 2§Ϙ
pv:

Row 4:

———————

page: b
page_location: 3
gmt: 2020-04-22
uv:
pv:

Row 5:

———————

page: a
page_location: 2
gmt: 2020-04-22
uv: 防 μX
pv:

Row 6:

———————

page: a
page_location: 3
gmt: 2020-04-22
uv: OAv□Q
pv:

Query UV based on page column

a2-test-clickhouse-1-223.sh :) select page,uniqMerge(uv) from ck_pv_uv_view group by page order by page\G

Row 1:
──────
page: a
uniqMerge(uv): 4

Row 2:
──────
page: b
uniqMerge(uv): 3

Row 3:
──────
page: c
uniqMerge(uv): 1

Query PV based on page column

```
a2-test-clickhouse-1-223.sh :) select page,sumMerge(pv)    from    ck_pv_uv_view group by page order by page\G


Row 1:
──────
page: a
sumMerge(pv): 8

Row 2:
──────
page: b
sumMerge(pv): 4

Row 3:
──────
page: c
sumMerge(pv): 1
```

The pre-computed binary data storage mode is similar to the figure below, and the total number of items is reduced by half, achieving the effect of "space" for "time".

NAME	PAGE	PAGE_LOCATION	CLICKS	GMT
甲	a	1	1	2020/4/22
甲	a	2	1	2020/4/22
甲	b	1	1	2020/4/22
乙	a	1	1	2020/4/22
乙	b	1	1	2020/4/22
乙	a	3	1	2020/4/22
乙	c	2	1	2020/4/22
丙	a	1	1	2020/4/22
丙	a	2	1	2020/4/22
丙	b	1	1	2020/4/22
丙	b	3	1	2020/4/22
丁	a	1	1	2020/4/22
丁	a	3	1	2020/4/22

PAGE	PAGE_LOCATION	UV	GMT
a	1	[a,1] value(sum:clicks):4 [甲,乙,丙,丁]	2020/4/22
a	2	[a,2] value(sum:clicks):2 [甲,丙]	2020/4/22
a	3	[a,3] value(sum:clicks):2 [乙,丁]	2020/4/22
b	1	[b,1] value(sum:clicks):3 [甲,乙,丙]	2020/4/22
b	3	[b,3] value(sum:clicks):1 [丙]	2020/4/22
c	2	[c,2] value(sum:clicks):1 [乙]	2020/4/22