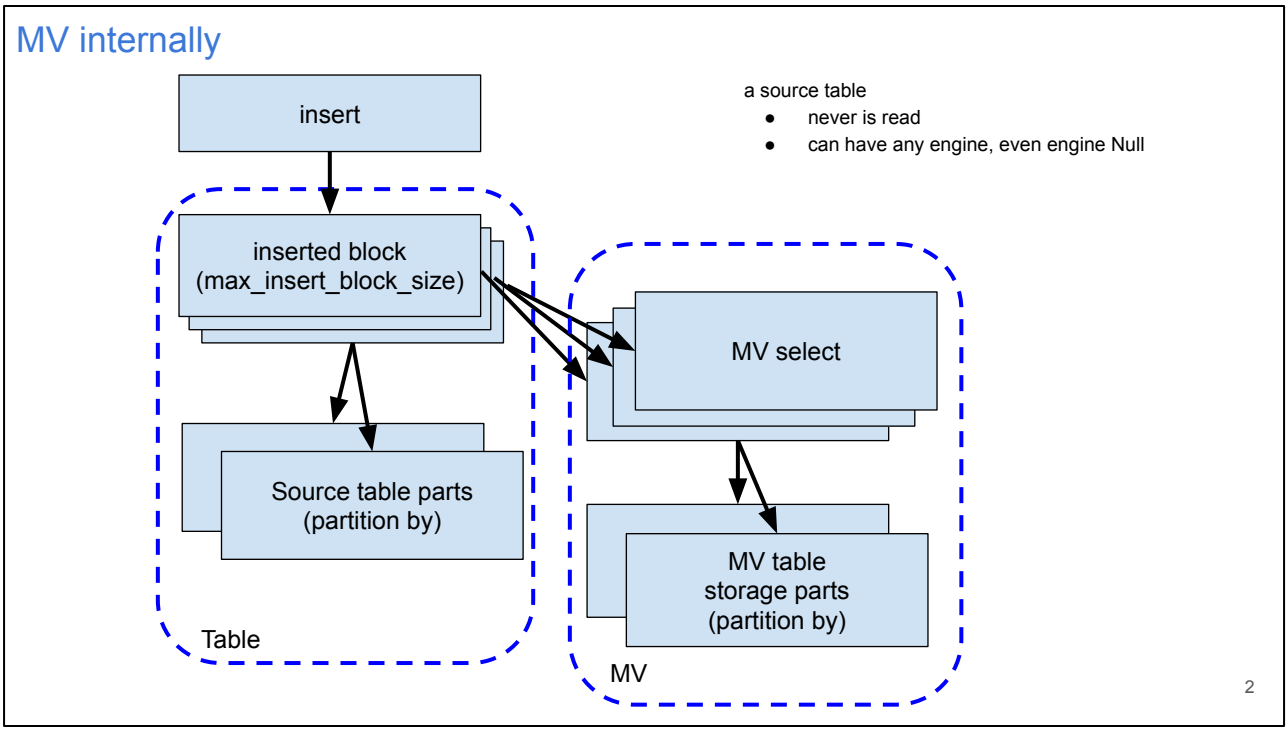
ClickHouse

Everything you should know about materialized views.

Denis Zhuravlev (Denny Crane), DBA, Revjet

I am Denis Zhuravlev a-k-a Denny Crane. I live in Atlantic Canada and I work at a company named Revjet (internet advertisement). I am a DBA and I have worked as a DBA for more than 20 years. Now I work mostly as Clickhouse and Vertica DBA.

Today I'll try to shed some light on Materialized View internals.



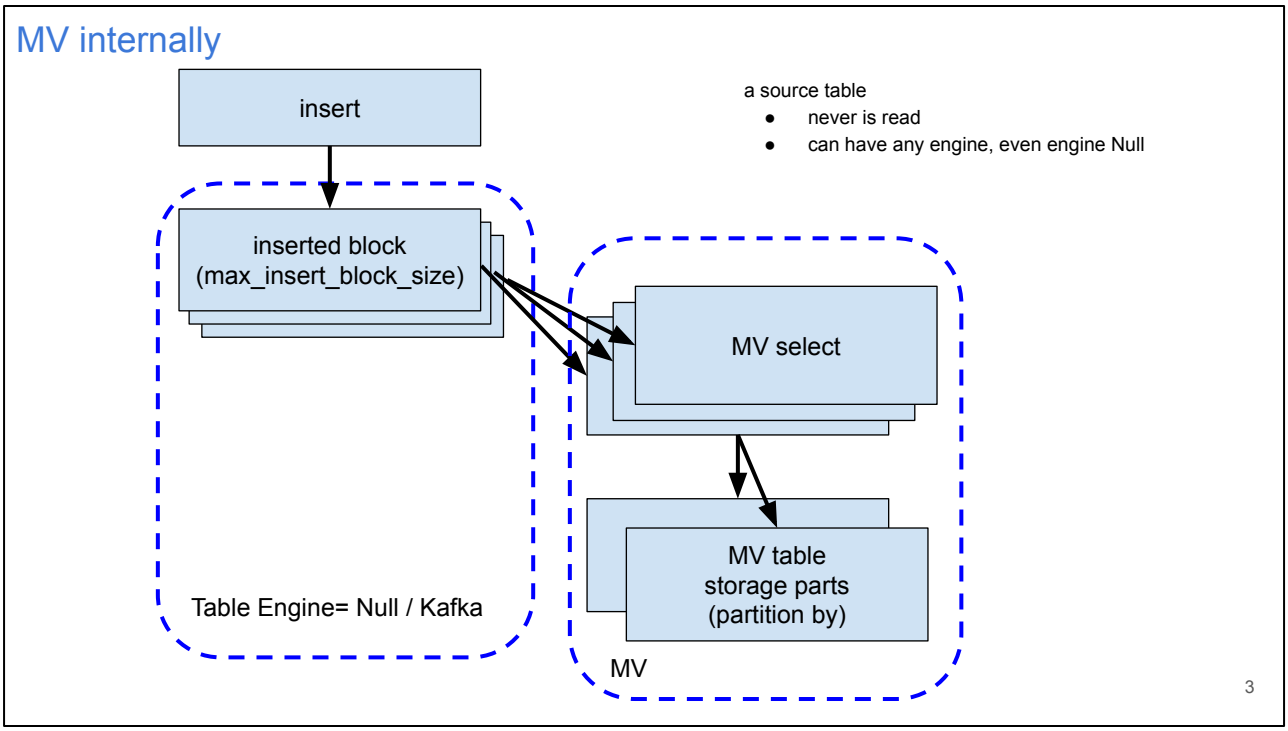
Probably you've heard more than once that an MV is an insert trigger. And that's true. The most tricky part is that an MV never reads its source table.

But how MV works internally? When CH gets an insert into a table with MV and for example this insert has 3 mil. rows. This Insert is processed in a streamable manner and its data is separated into 3 blocks because of settings -- in particular because of max_insert_blocks_size.

Then these blocks will be written into a table and every block can form multiple parts because of a table partitioning. For example a table has a partitioning expression and if an inserted block contains multiple partitioning key values then it will be written as multiple parts.

If this table has an MV then MV will get this block and not FROM the table but from the insert. Then MV select will be processed over the inserted block.

So our insert brought 3 mil. rows and the MV select was triggered 3 times.
And for every select's result MV stored this result into an mv storage table.
If this storage table is partitioned then this result might be separated into several parts as well.



Important part: MV never reads the source table (except for the populate stage). And the source table can have any Engine.
Engine type does not matter. It can even be Engine Null or Kafka. They both don't store data themselves.

What is going on in the case of Kafka or Null ? When an insert comes, the data is not written anywhere because Kafka and Null do not store inserted data.
But the MV select happens against the inserted block and the select's result is written into MV storage.

MV use cases

- Data pre-aggregation (Summing/Aggregating)
- Entry-point / Extract-Transform
- Duplicate data to another table with different PK
  (+emulation of inverse indexes)
- Kafka

And how can we use MV?

- For example it can be used to pre-aggregate data. In this case data is usually stored in SummingMT or AggregatingMT. For example I have a table with data for the last 3 years and 400 bil. rows. I also have several pre-aggregated tables with the same data that are filled using MVs. Data in these tables is grouped by hour/day/month and the number of rows is millions instead of billions.

- Also I use MV as a sort of entry point and extract-transform tool. I have a Null table and have a script which knows how to insert data from several different data sources into this Null table. It's sort of one-column data with a single String column and data is JSONs. This Null table has multiple MVs and they perform different data transformations and write results into different destinations.

- Also you can duplicate data using MV and store all rows in another table sorted or partitioned by other columns for faster search.
  I use MV to emulate inverted-index. It's pretty much the same idea. I have a table with 500 columns. This table has a high cardinality column transactionid but it's sorted by other columns. So I made an additional narrow table that

stores primary key columns from the main table and this transcationid column and sorted this table by transactionid. So we do lookups by index by that transaction id from the narrow table and jump by primary key into the main table using subqueries. And this additional table is filled by MV.

- Also MV is used with Kafka or RabbitMQ Engines. This is the only way to store data from these engines.

MV with TO vs MV .inner

Implicit table `.inner.mv1`

```
CREATE MATERIALIZED VIEW mv1
ENGINE = SummingMergeTree
PARTITION BY toYYYYMM(d)
ORDER BY (a, b, d)
AS
SELECT a, b, d, count() AS cnt
FROM source
GROUP BY a, b, d;
```

```
DESCRIBE TABLE `.inner.mv1`

┌─name─┬─type───┬─default_type─┐
│ a    │ Int64  │              │
│ b    │ Int64  │              │
│ d    │ Date   │              │
│ cnt  │ UInt64 │              │
└──────┴────────┴──────────────┘
```

Explicit table dest

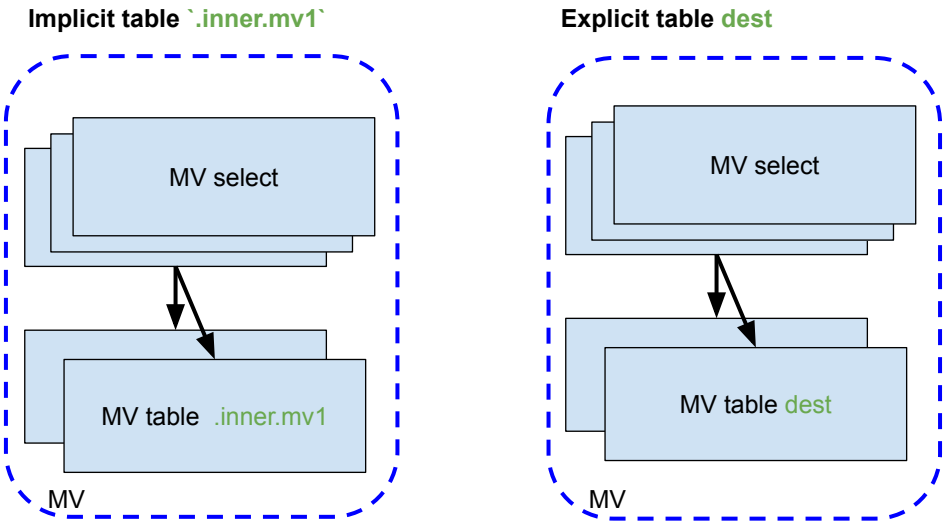
```
CREATE TABLE dest
(a Int64, b Int64, d Date, cnt UInt64)
ENGINE = SummingMergeTree
PARTITION BY toYYYYMM(d)
ORDER BY (a, b, d);

CREATE MATERIALIZED VIEW mv1
TO dest
AS
SELECT a, b, d, count() AS cnt
FROM source
GROUP BY a, b, d;
```


How MV stores its data

MV uses an ordinary table to store its data.
We can use two syntaxes to define the storage table. Syntax with Engine and with TO.
If you use syntax with Engine then MV creates an implicit table and names this table as dot-inner-dot-MV-name. Such ‘inner’ table has a dot in the beginning of the name so it can be queried only using backticks or double-quotes. It’s a restriction for objects with a dot in the name.
Inner table is an ordinary table. You can use commands ‘describe’/’show create table’. You can insert data into this table directly or query it using select or alter or truncate. It's just a table.

I don’t use this syntax with Engine. I use only TO syntax. It’s more convenient.
I create a storage table explicitly. It’s the same table as inner.
After that I create MV and point MV to my explicit table using TO syntax.

MV with TO vs MV .inner



6

What is going on under the hood?

MV executes Select against an inserted block and stores result into inner table.

In case of TO MV executes Select against inserted block and stores a result TO ‘dest’ table.

When TO syntax is used it does not create ‘inner/ table. The data is stored only once in the ‘dest’ table.

MV with TO vs MV .inner

Implicit table `.inner.mv1`

- optimize_move_to_prewhere does not work
  should query from `.inner.mv1`
- populate does work
- drop table mv1 implicit .inner table is removed

Explicit table dest

- populate does not work (use insert)
- drop table mv1 does not remove TO table (dest)

What else is different for implicit inner and explicit TO tables?

optimize_move_to_prewhere does not work with MV. When you query MV by name this optimization does not work. You have to query ‘inner’ table to get full optimization. This is inconvenient.

Populate is supported with inner table syntax and is not supported for TO syntax.

Drop table MV drops inner table as well. In case of drop MV the explicite table stays which is convenient.

MV with TO vs MV .inner

- insert into mv1 / insert into `.inner.mv1` / insert into dest

What's important?
You can insert into MV, into .inner into dest (TO) CH executes an ordinary insert.
MV select does not participate.

Populate

- I never use it

9

Poplulate

I never use it.
It’s unusable now.
During the populate stage MV misses rows inserted into its source table because MV will be created after the populate finished. It can be addressed and will be addressed eventually.

Another problem that populate runs against all data. If you have a billion rows in a source table then most probably the populate will be interrupted by a timeout or by out of memory.

I’ll show today how to fill MV manually without using populate.

Common mistakes

- Storage table knows nothing about MV and about MV's select !!!

Common errors and questions about MV

First or all. A storage table where MV data is stored knows nothing about MV or about your SELECT in MV or about the source table.

For the storage table insert by MV is just an insert nothing special.

The storage table 'inner' or 'dest' is independent and merges its data according to its own ORDER / PARTITION by rules.

Common mistakes

Incorrect PK (order by) of storage-table

Incorrect

```
CREATE MATERIALIZED VIEW mv1
ENGINE = SummingMergeTree
PARTITION BY toYYYYMM(d)
ORDER BY (a, b)
AS SELECT a, b, d, count() AS cnt
FROM source
GROUP BY a, b, d;
```

Engine rules:

```
a -> a
b -> b
d -> ANY(d)
cnt -> sum(cnt)
```

Correct

```
CREATE MATERIALIZED VIEW mv1
ENGINE = SummingMergeTree
PARTITION BY toYYYYMM(d)
ORDER BY (a, b, d)
AS SELECT a, b, d, count() AS cnt
FROM source
GROUP BY a, b, d;
```

Engine rules:

```
a -> a
b -> b
d -> d
cnt -> sum(cnt)
```


The most common mistake is the wrong ORDER BY in the storage table.
In this example I create a SummingMT table partitioned by month and ordered by columns and A and B.

And I use select with A, B, D columns. Such MV will be filled with garbage because column D will be corrupted by SummingMT.
SummingMT aggregates all columns not mentioned in ORDER BY section by SUM or ANY. SUM is for numeric and ANY is for other types.
The column D has a Date type and it will be aggregated by ANY. And during populate you may see something close to the correct result but later merges will destroy data in the D column.

The common rule is that ORDER BY of the storage table has to be the same as GROUPBY in SELECT.

Common mistakes

Double grouping (incorrect)

```
CREATE MATERIALIZED VIEW mv1
ENGINE = AggregatingMergeTree
PARTITION BY toYYYYMM(hour)
ORDER BY hour
POPULATE
AS
SELECT toStartOfHour(time) hour,
       maxState(cnt_by_minute) max_by_hour,
       sumState(cnt_by_minute) sum_by_hour
FROM
(
    SELECT minute, count() AS cnt_by_minute
    FROM source
    GROUP BY minute
)
GROUP BY hour
```

```
insert into source
values (now()), (now())

result: max_by_hour = 2
```

```
insert into source
values (now());
insert into source
values (now());

result: max_by_hour = 1
```


The second common mistake is double-groupings in MV selects. They do not work.
Because MV select works against the inserted block not against the source table.
And results may differ if rows came in several or in one inserts.
Again it can work during populate but cannot during inserts later.
In this example I compute sumState by hour from count by minute and maxState by hour from count by minute.
Obviously sumState CAN work but maxState CANNOT.

If we insert 2 rows together count by minute will be 2, max by hour will be 2.

If we insert rows one by one, count by minute will be one for both inserts SO max by hour will also be one.

And sum by hour will be correct which makes more confusion.

Common mistakes

MV is not related to the source table !!!

- MV does not watch for Replacing/CollapsingMT of the source table
- MV knows nothing about truncate / alter delete / alter update / drop partition / drop table / rename of the source table
- MV can store data for different period

Important: MV is not related to the source table.

It has zero knowledge about source Replacing or Collapsing MT tables and cannot follow their merges.

It’s simply impossible because MV acts as an insert trigger.

MV has zero knowledge about truncates or deletes or drop or rename of / in the source table.

And this is good. MV is able to store data for a different period.
The source table for example has data for the last 6 months and MV can store aggregated data for the last 10 years.

Common mistakes

MV uses column's names !!!

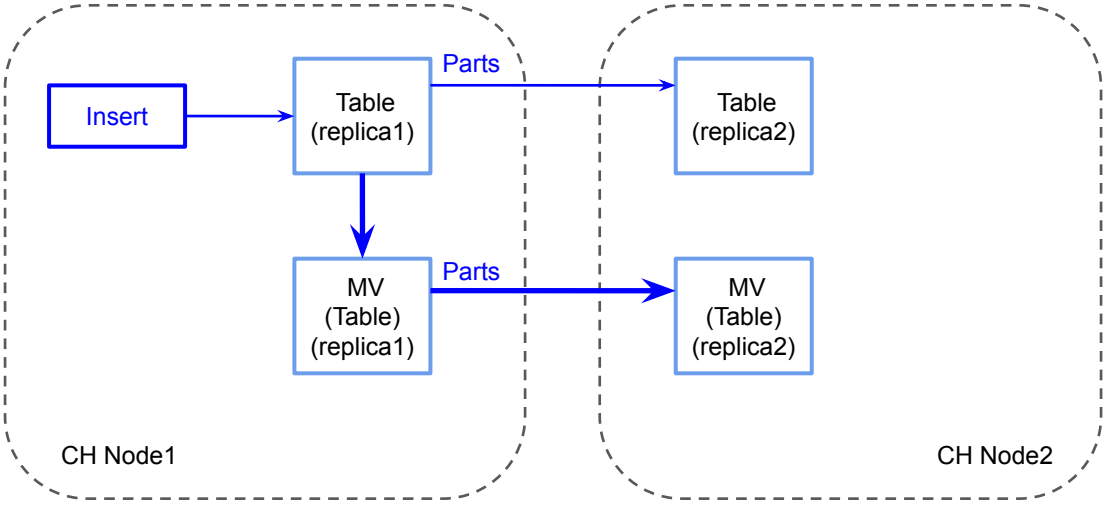
```
CREATE MATERIALIZED VIEW
mv1 (a Int64, d Date, cnt Int64)
ENGINE = SummingMergeTree
PARTITION BY toYYYYMM(d)
ORDER BY (a, d)
POPULATE
AS
SELECT a, d, count()
FROM source
GROUP BY a, d;
```

```
CREATE MATERIALIZED VIEW
mv1 (a Int64, d Date, cnt Int64)
ENGINE = SummingMergeTree
PARTITION BY toYYYYMM(d)
ORDER BY (a, d)
POPULATE
AS
SELECT a, d, count() as cnt
FROM source
GROUP BY a, d;
```


The next problem.
MV uses column's names. You should use aliases in MV select to match the column names of the destination table.
In this example you can see that MV has a column named **cnt**. But the SELECT count() function does not have an alias. And count goes nowhere.
And the most ugly thing is that at the populate stage CH uses columns' positions and this MV will be filled with correct values during populate and with 0 in CNT column during inserts.
This is a known issue and will be fixed eventually.

And this behavior appeared recently because now CH allows different column's set for a MV select section and a MV storage table. Now we can have columns with defaults in a MV storage table. And MV SELECT can query more columns than its storage table has.

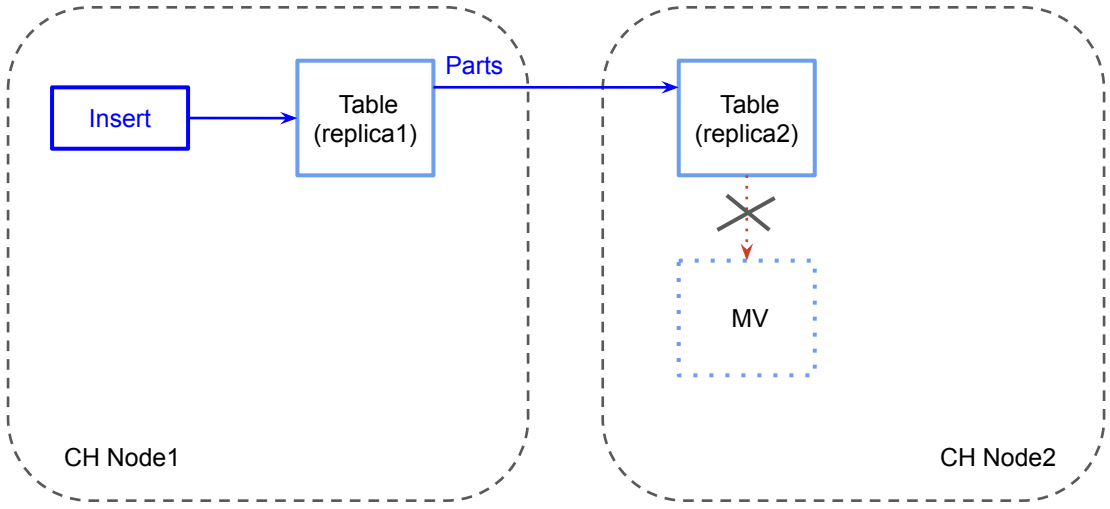
ReplicatedMT & MV



15

Now I'll explain how MV can be used with Replicated Engines.
On this slide INSERT creates a part in the source table and this part is replicated and how we can replicate MV data?
We should create an MV storage table with a Replicated Engine with the same shard and replica arguments as in the source table and CREATE the same MV on a replica.
And parts which are created in MV because of inserts will be replicated the same way as the source table data.
It's a simple replication for MV storage tables.

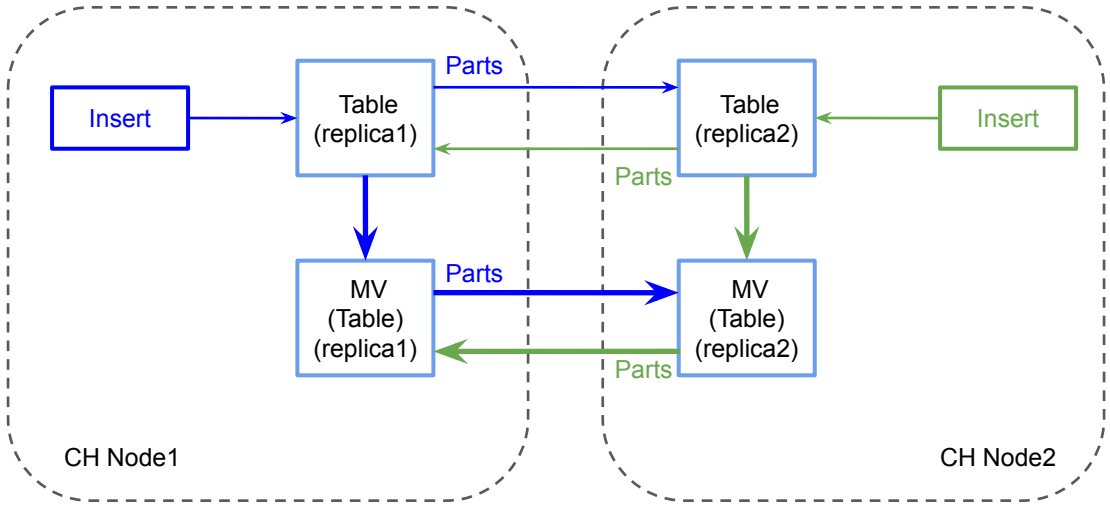
ReplicatedMT и MV



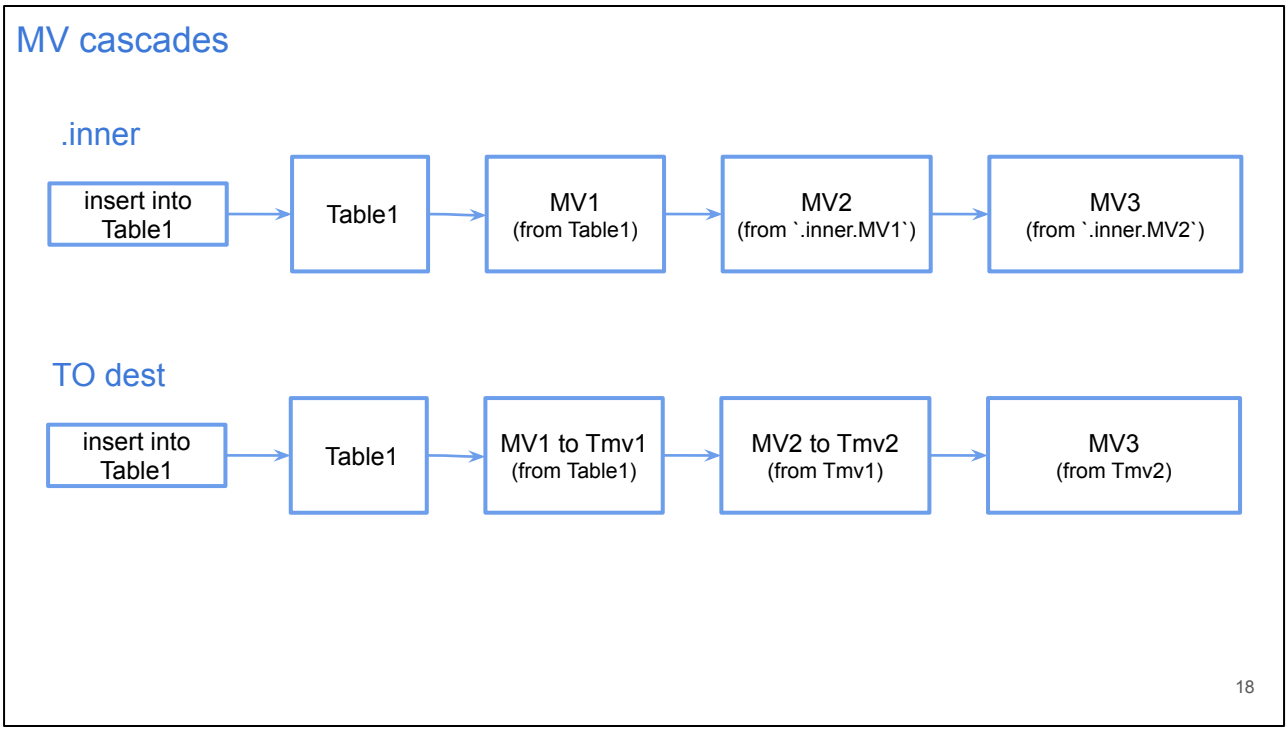
People often ask: Can I insert on one server and create a MV for the source table replica on another node?
The answer is: No, it’s impossible.
Again MV is an insert trigger and MV works on a node where inserts happen.

Replication works with parts and does not involve inserts. Replication sends files , not a log with queries.

ReplicatedMT и MV



The most common schema is to create the same tables and the same MV on all replicas and insert on one or multiple replicas.
Parts will be the same on all replicas for both a source table and MV and everything will be consistent.

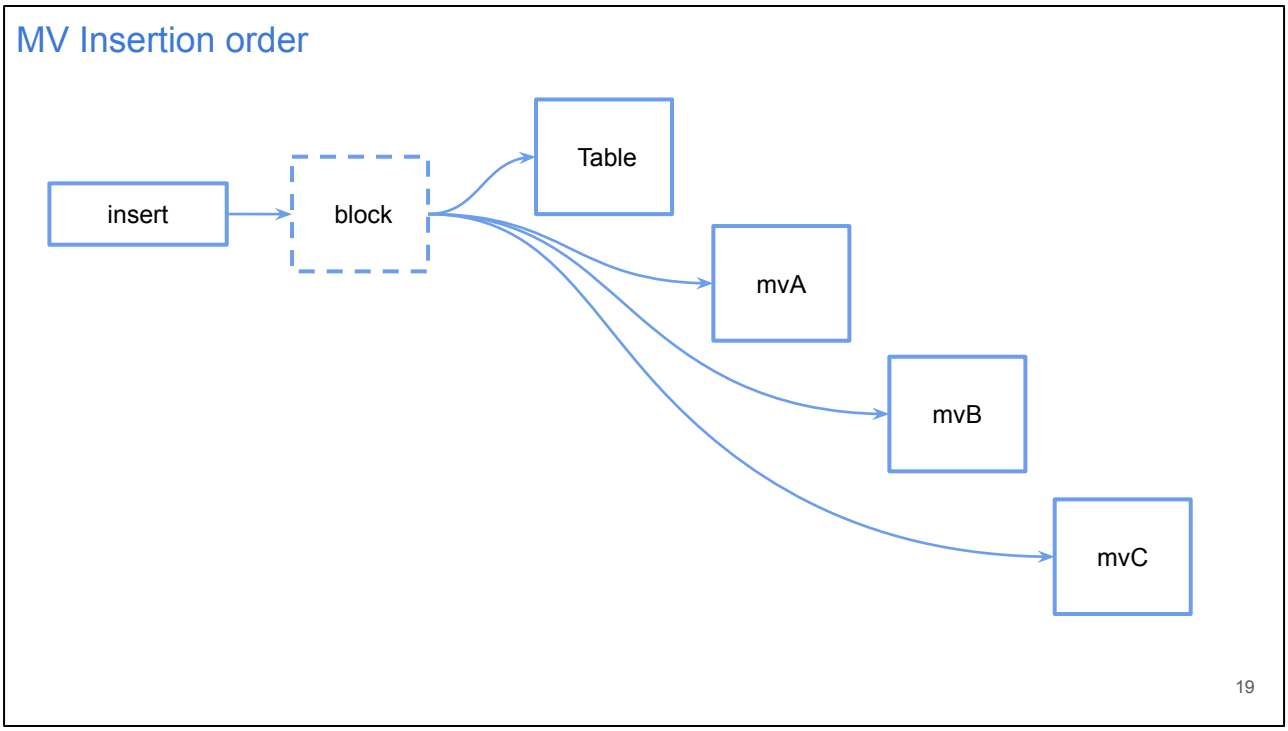


Cascades. Starting from a CH version 19.16 cascading was improved.
And we can create a MV for another MV. It usually is used if the first MV has some hard computation so the second MV can use result from the first MV.

There is only one annoying issue: It needs to use a storage table as a source for the second MV.
So for MV with implicit storage use inner table and for MV with explicit storage use TO destination table in the FROM section of the next MV.
When the first MV inserts data into its storage table it triggers the next MV.
This issue will be addressed eventually I hope.

And worth to mention that before the CH 19.16 it was exactly the opposite; we had to use MV name in the next MV and could not use storage tables. It brings a lot of confusion during upgrades.

But because of this change MV cascades are now supported for Kafka engines.



Can we create multiple MVs for a table? Yes we can.
I don't recommend to create 50 MVs (it's a real case of one user who had problems with insertion performance) because if you have 50 MVs during insert it will create at least 51 parts obviously it causes a lot of random I/O and works badly on HDD.

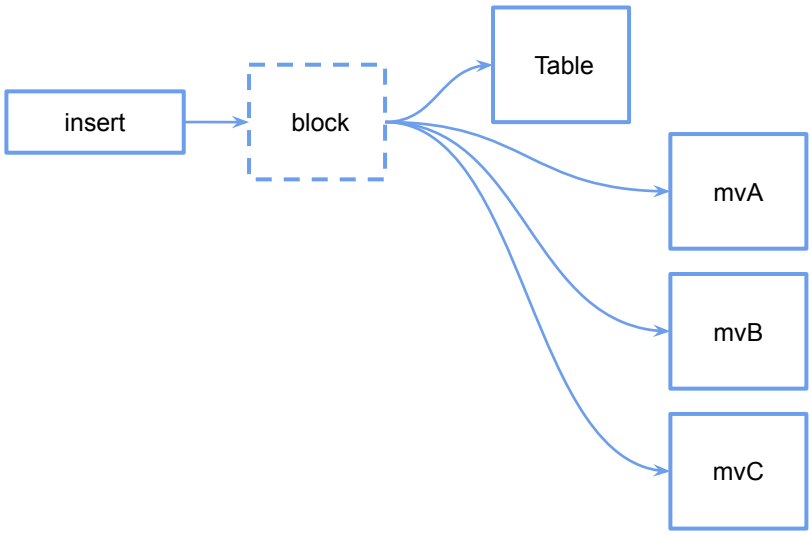
How does an insert work in case of multiple MV ? Insert creates a block. The block forms a part and the part is written into the source table.

Then the block is propagated into the first MV and creates a part there.

Then the same block is propagated into the second MV and so on.

MVs are processed in the alphabetical order.

MV Insertion order & parallel_view_processing

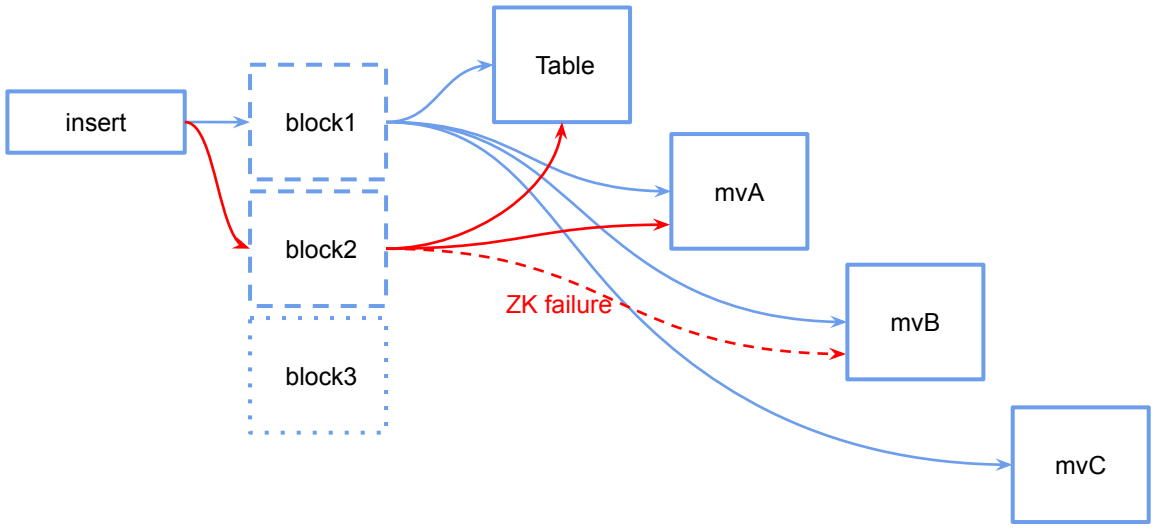


There is a setting parallel_view_processing.

If this user setting is enabled (equal 1) then right after an insert into the source table is completed (part is written) MVs are processed in parallel by multiple threads.

This setting can speed-up inserts if you have more than 2 MVs.

MV Insertion order



21

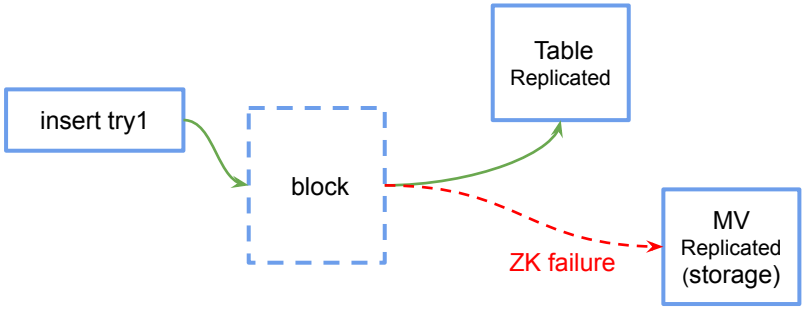
Why do we need to know about MV processing order?
Because it is needed to get consistent data even with insert failures.
CH inserts are not atomic among a source table and MVs.

In this slide CH got an insert with 3 mil. rows. This insert was separated into 3 blocks.
The first block was processed successfully and the source table and all 3 MVs got this block.
While the second block was processed an error happened during insertion into the second MV B. For example a network error caused a ZK error.
The outcome of this error is inconsistent state of the MVs.
The source table got 2 blocks, the first MV A got 2 blocks.
MV B and MV C got 1 block.

The 3-d block never happened, the client got an exception and did not send it.

So we got inconsistent data

MV insert deduplication

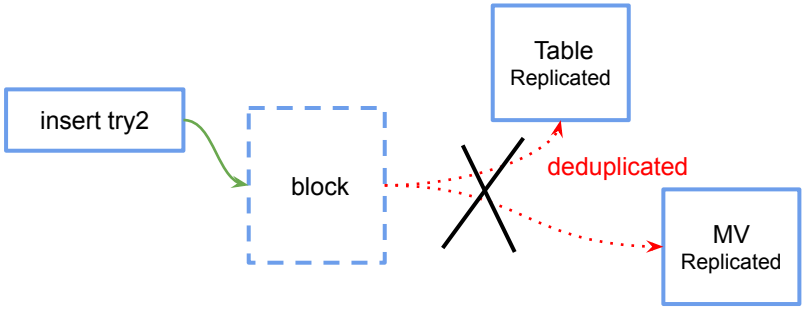


But CH has an important feature insert_deduplication. It works now only for Replicated engines.
Insert into the Replicated table can be repeated safely because the repetition will be deduplicated.
It is implemented to be able to retry inserts in case of an error or a timeout.
The last 100 inserts can be repeated but retried inserts must be completely identical. The same rows in the same order. Because deduplication is based on a checksum of a block.

In this slide we have a Replicated Table and MV storage is also Replicated. Insert happened and created a part for a source table and a checksum for this block was written into ZK.

During insertion into MV an error happened and MV did not create a part.

MV insert deduplication



- setting deduplicate_blocks_in_dependent_materialized_views = 0 (default)

24

We can do a retry and repeat exactly the same insert. But it will be deduplicated by the source table because a checksum for this block already exists in ZK.

Replicated MV gets no data in this case because by default MV inherits deduplication from the source table.
This behavior is one of the reasons for data inconsistency. But it can be changed.
If we enable this parameter then MV will start to check deduplication by itself using its own checksums. And MV will get data on retry.

This setting is disabled by default because very often MV storage tables like Summing or AggregatingMT are very narrow and can form the same blocks for different inserted data. Imagine we always insert 1000 rows and in this MV we store the number of inserted rows. Just a number 1000.
Then for 2 different row sets inserted into the source table we insert exactly the same row into MV with the number 1000. We do not want the next insert to be deduplicated by MV.

What we can do about it? If you experience such issues with Summing or AggregatingMT I recommend to add a column into the source table with an insert unique identifier and propagate this column into MV but don't include this column into orderby section of MV storage.

In this case you can enable this parameter but inserts into MV will not be falsely deduplicated (because of this uniq id).
And this column will be collapsed during merges so it will not take much disk space. After some time this column can be cleared by TTL or ‘alter table clear column’ command.

JOIN & MV

- Multiple blocks -> multiple JOIN
- External dictionaries и dictGet or engine=Join и joinGet

Joins And MV
I don’t recommend using JOINs in MV because they slowdown inserts dramatically. CH uses a HashJoin algorithm and it requires to build a hash table in memory for the right table. And this hash table must be built for each block for every insert. It slows down inserts and wastes memory in case of parallel inserts.

Instead of Joins I use external dictionaries and dictGet or Engine=Join and joinGet. In this case we have a pre-built hash table in memory. And this hash table is shared for all inserts.

JOIN & MV

```
CREATE MATERIALIZED VIEW mv1
...
AS
SELECT ...
FROM
(
     SELECT ...
     FROM source join some_dimension on (...)
)
```


If despite my recommendation you decide to put a join into MV you need to know which table will trigger the MV.
So MV searches for the first real table in the From section and triggers on inserts into this table only.

JOIN & MV

- MV for both tables JOIN (2 MV to dest + RIGHT JOIN)

```
CREATE MATERIALIZED VIEW mv1
TO dest
SELECT ...
FROM source left join some_dimension on (...)


CREATE MATERIALIZED VIEW mv2
TO dest
SELECT ...
FROM some_dimension right join source on (...)
```


If you need to trigger MV on insert into both tables you should create 2 MV with the same TO destination table.

In the second MV you need to swap tables and make the other table the first in from section.

In this example I replaced leftJOIN with RIGHTJOIN to achive that.

MV from distributed / To distributed

- From Local* to Local
- From Distributed to Distributed
- From Local* to Distributed
- From Distributed to Local

* (Local is behind of Distributed it does not matter)

Another frequent question is how to use MV in cluster with distributed tables.

All 4 options are possible.
The most used is the first option. This option From local to Local is the most used and the most efficient. Inserted data is processed locally without network overhead.

Other options should be used only if you really need them.
For example the second is useful if you need to shard data in MV by another sharding key. Let's say a source table is sharded by transaction identifier but MV has to be sharded by user identifier.

The third option is almost the same. You insert directly into Local table and want MV to shard ITS data somehow.

4-th option is also possible but I can’t imagine the use case.

* it does not matter that the local table is behind the Distributed table. Distributed table propagates inserts into Local and MV over Local catches inserts.

MV from distributed / To distributed

```
CREATE TABLE src (A Int64, B String) Engine=Memory;

CREATE TABLE src_log (A Int64, B String) Engine=File(JSONEachRow);

CREATE MATERIALIZED VIEW src_log_mv TO src_log AS
SELECT A, B FROM src WHERE A % 2 = 0;

INSERT INTO src SELECT number A, 'x' B FROM numbers(4);
INSERT INTO src SELECT number A, 'y' B FROM numbers(4);

SELECT count() FROM src_log
┌─count()─┐
│       4 │
└─────────┘

sudo wc -l /var/lib/clickhouse/data/default/src_log/data.JSONEachRow
4 /var/lib/clickhouse/data/default/src_log/data.JSONEachRow
```


Actually you can use anything in the TO section of MV.
For example engine File

MV from distributed / To distributed

```
SERVER2: CREATE TABLE dest (A Int64, B String)
Engine=MergeTree ...;

CREATE TABLE remote_src_log AS remote('SERVER2', default, dest)

CREATE MATERIALIZED VIEW src_log_mv1
TO remote_src_log AS
SELECT A, B
FROM src WHERE A % 2 = 0;
```


or you can create a remote table and insert data into a table in another cluster.

How to alter MV

Implicit table (.inner.mv1)

1. stop inserts
2. detach table mv1
3. alter table `.inner.mv1`
4. attach materialized view mv1

4a. edit .sql + attach table

Explicit table (TO dest)

1. stop inserts
2. alter table dest
3. drop table mv1
4. create materialized view mv1

Can MV be altered? Yes. It is possible. You can change the storage table and then change the MV select section.

In case of an implicit ‘inner’ table you need to stop insertion.
Then Detach MV.
If you did not stop insertion before detach your MV will miss inserts.
After that alter the storage table as usual. Add columns / modify columns.
Then attach MV with a new select.

In case of an explicit TO (dest) table you need to stop insertion.
Then Alter the dest table.
Then Drop MV/ Create MV.
That is one of the reasons why I use only explicit storages (MV with TO). It is easier and clearer to maintain.
MV name does not matter in this case.
In queries I use the name of the dest table. It’s much more transparent.

How to alter MV

Implicit table (.inner.mv1)

```
DETACH TABLE mv1

ALTER TABLE `.inner.mv1`
    ADD COLUMN b Int64 AFTER a,
    MODIFY ORDER BY (a, b)

ATTACH MATERIALIZED VIEW mv1
ENGINE = SummingMergeTree
ORDER BY (a, b) AS
SELECT a, b, sum(amount) AS s
FROM source
GROUP BY a, b
```

Explicit table (TO dest)

```
ALTER TABLE dest
    ADD COLUMN b Int64 AFTER a,
    MODIFY ORDER BY (a, b)

DROP TABLE MV1

CREATE MATERIALIZED VIEW mv1
TO dest
SELECT a, b, sum(amount) AS s
FROM source
GROUP BY a, b
```


How does it look from SQL view?
I do detach table MV.
Do alter. Here is a complicated case: I am adding a new dimension to a table with SMT, I add a new column and I modify the table's order_by in one single query. It does not change anything on a disk but it changes merge behavior.
Attach MV with a new select.

In case of an Explicit table it is much easier. We change the storage table and drop/create MV.

How to alter MV

```
SET allow_experimental_alter_materialized_view_structure = 1;

ALTER TABLE mv1 MODIFY QUERY
     SELECT v * 2 as v, 1 as v2
     FROM source;
```


The only inconvenience that inserts must be stopped
But there is an experimental feature.
This feature allows to change MV select on the fly.
You can alter the storage table before or after this alter.

How to fill MV for existing data (populate)

- create MV with where date >= date_in_future
  wait for date_in_future, insert all before date_in_future

```
CREATE TABLE dest(a Int64, d Date, cnt UInt64)
ENGINE = SummingMergeTree
PARTITION BY toYYYYMM(d) ORDER BY (a, d);

-- create MV c where date >= in_the_future
CREATE MATERIALIZED VIEW mv1 TO dest AS
SELECT a, d, count() AS cnt
FROM source
WHERE d >= '2020-11-01'
GROUP BY a, d;

-- arrives 2020-11-01
INSERT INTO dest -- insert all for before in_the_future
SELECT a, d, count() AS cnt
FROM source
WHERE d < '2020-11-01' -- piece by piece by 1 month (or .. day)
GROUP BY a, d;
```


How we can populate MV manually for a huge table with billions rows? There are several ways.

Probably your source table has a column with a timestamp.
We can pick a date in the future. And create an MV with this date in where section.
After this date arrives you can fill MV with data for dates before this future date (already past)

How to fill MV for existing data (populate)

- create MV / insert all before today , (before current partition beginning)
- wait for a new month (new partition)
- drop incomplete partition, re-compute it

```
CREATE TABLE dest(a Int64, d Date, cnt UInt64)
ENGINE = SummingMergeTree
PARTITION BY toYYYYMM(d) ORDER BY (a, d);

CREATE MATERIALIZED VIEW mv1 TO dest AS
SELECT a, d, count() AS cnt
FROM source
GROUP BY a, d;

INSERT INTO dest -- insert all for before today
SELECT a, d, count() AS cnt
FROM source
WHERE d < '2020-10-01' -- piece by piece by 1 month (or .. day)
GROUP BY a, d;

-- arrives 2020-11-01
ALTER TABLE dest DROP PARTITION 202010; -- because it is inconsistent

INSERT INTO dest
SELECT a, d, count() AS cnt
FROM source
WHERE d >= '2020-10-01' and d < '2020-11-01'
GROUP BY a, d;
```


Another way.

You create a MV on the fly.
The current partition will be incomplete.
When a new partition starts you can drop the incomplete partition and populate it from the source table.

Or you can use alter delete and delete incomplete day.

How to fill MV for existing data (populate)

- stop inserts / insert all for today
- create MV
- start inserts / insert all before today

```
CREATE TABLE dest(a Int64, d Date, cnt UInt64)
ENGINE = SummingMergeTree
PARTITION BY toYYYYMM(d) ORDER BY (a, d);

-- stop inserts
INSERT INTO dest -- insert all for today
SELECT a, d, count() AS cnt
FROM source
WHERE d >= today()
GROUP BY a, d;

CREATE MATERIALIZED VIEW mv1 TO dest AS
SELECT a, d, count() AS cnt
FROM source GROUP BY a, d;

-- start insert
INSERT INTO dest -- insert all for before today
SELECT a, d, count() AS cnt
FROM source
WHERE d < today() -- piece by piece by 1 month (or .. day)
GROUP BY a, d;
```


the third way.

We can stop inserts.
Create MV.
Fill the storage table with today's data.
Start inserts.
Insert the rest of the data for before today.

How to fill MV for existing data (populate)

- snapshot / alter table freeze

```
CREATE TABLE dest(a Int64, cnt UInt64)
ENGINE = SummingMergeTree ORDER BY (a);

-- stop inserts
ALTER TABLE source FREEZE;

CREATE MATERIALIZED VIEW mv1 TO dest AS
SELECT a, count() AS cnt
FROM source GROUP BY a;

-- start inserts
create table temp_snapshot as source;

-- mv .../shadow/1/data/default/source/* .../data/default/temp_snapshot/detached/
-- cd .../data/default/temp_snapshot/detached/
-- let i=1;for f in `ls -1` ; do echo $i $f;((i++)); echo "alter table temp_snapshot
attach part '$f';"|clickhouse-client ; done

INSERT INTO dest -- insert into mv storage from temp_snapshot
SELECT a, count() AS cnt
FROM temp_snapshot -- piece by piece by 'a'
GROUP BY a;
```


One more using snapshots / freeze

For example you don't have a date column.
Create a destination / MV storage table
stop insert
Freeze the source table -- is very quick and cheap operation
Create MV
start insert
Create a table with the same structure as the source table
attach parts from freeze into this temp
insert into dest from this temp table

You can drop this table later.

This temp table does not use disk space because of hardlinks

cookies

Alter MV and SummingMergeTree
https://gist.github.com/den-crane/3a8d57253711e277b2a60a75b5dfeef6

How to convert not replicated MV to replicated
https://gist.github.com/den-crane/80cb95e74f046be2d8ffae58d9b04e8f

How to alter MV
how to convert MV with implicit storage .inner to explicit storage (with TO)
https://gist.github.com/den-crane/431010ca08b9e51b960e55344b1dbbe3

MV_populating_with_freeze
https://gist.github.com/den-crane/64c05be67ef8a926242011188ce96f44

Several_MV_one_internal_store
https://gist.github.com/den-crane/005633470c70877dd28c00211cd9fcfb

Populate AggregatingMergeTree through null table
https://gist.github.com/den-crane/f7382cd4f1f859ff6ac46afe7dc9925a

Rename MV in CH before 19.8.3.8
https://gist.github.com/den-crane/d9a0bceb58a7d5314ea66e8edbebc9f7

I recommend to play with these examples to get used to it and get understanding of what you can do.