Data Modeling in NoSQL (C*) - Advanced

Big Data Systems

Dr. Rubi Boim

Happens to the best

- In 2019 Jennifer Aniston joined Instagram and posted a single photo
- 1m followers after 5 hour and 16 minutes from registering
  world record
- More than 7m follower (24 hours)
- More than 9m likes for that photo (24 hours)
- Instagram crashed temporarily

Previously we learned

- Each query should be satisfied by one partition
  denormalization…

videos_by_genre	
genre	K
release_date	▼
video_id	▼

videos_by_id	
video_id	K
release_date	
title	
rating	
duration	
{genres}	

Previously we learned

- Each query should be satisfied by one partition

  denormalization…

videos_by_genre	
genre	K
release_date	▼
video_id	▼

videos_by_id	
video_id	K
release_date	
title	
rating	
duration	
{genres}	

```
SELECT video_id
FROM videos_by_genre
WHERE genre = “action”
```

↓

```
for (video : result){
    SELECT *
    FROM videos_by_id
    WHERE video_id = video
}
```

How many queries can this generate?

Previously we learned

- Each query should be satisfied by one partition
  denormalization…

videos_by_genre	
genre	K
release_date	▼
video_id	▼

videos_by_id	
video_id	K
release_date	
title	
rating	
duration	
{genres}	

videos_by_genre	
genre	K
release_date	▼
video_id	▼
title	
rating	
duration	

Previously we learned

- Each query should be satisfied by one partition
  denormalization…

videos_by_genre	
genre	K
release_date	▼
video_id	▼

videos_by_id	
video_id	K
release_date	
title	
rating	
duration	
{genres}	

```
SELECT *
FROM videos_by_genre
WHERE genre = "action"
```

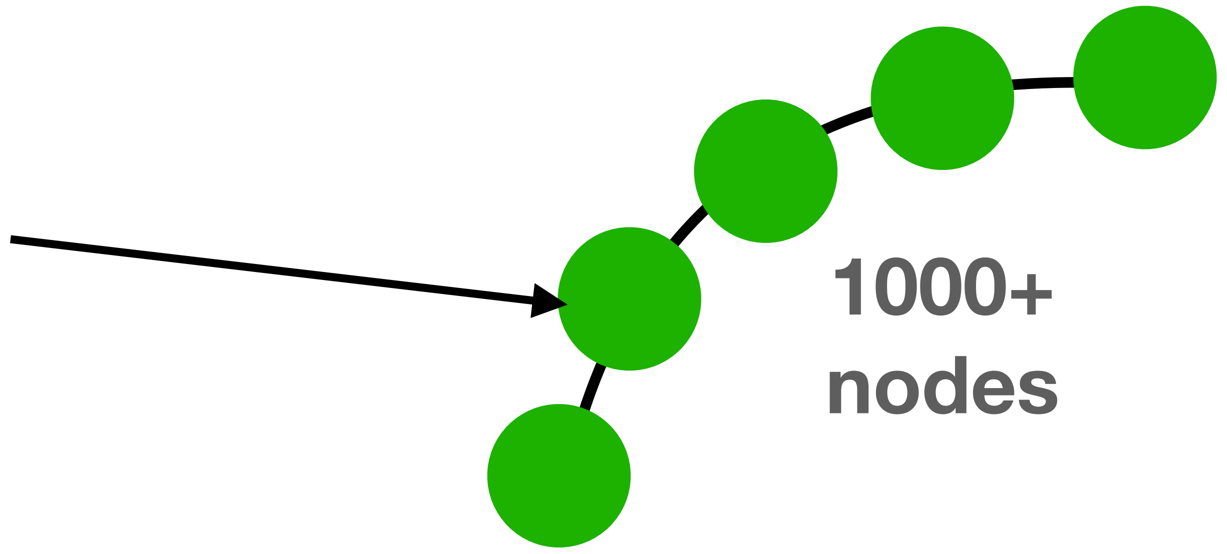
We add ("duplicate") all the attributes we need for the query

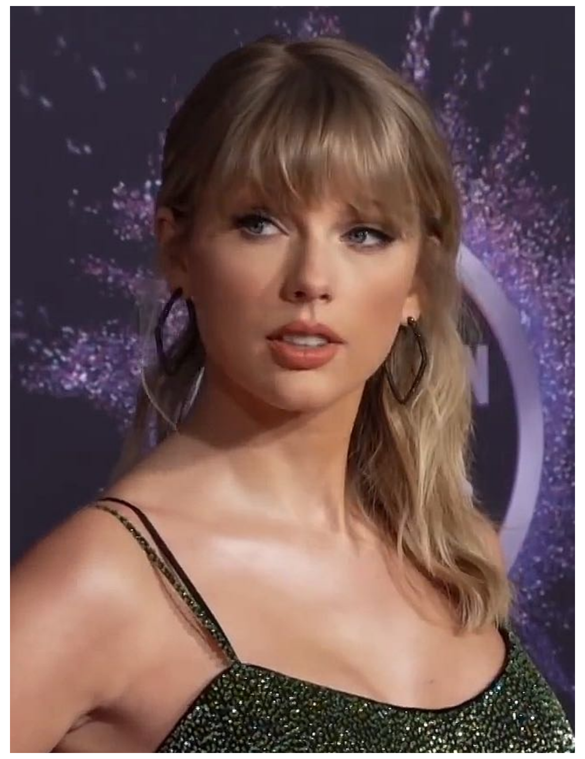
videos_by_genre	
genre	K
release_date	▼
video_id	▼
title	
rating	
duration	

But what happens if the partition is “large”

- There can be more than 10m rows in this partition

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	





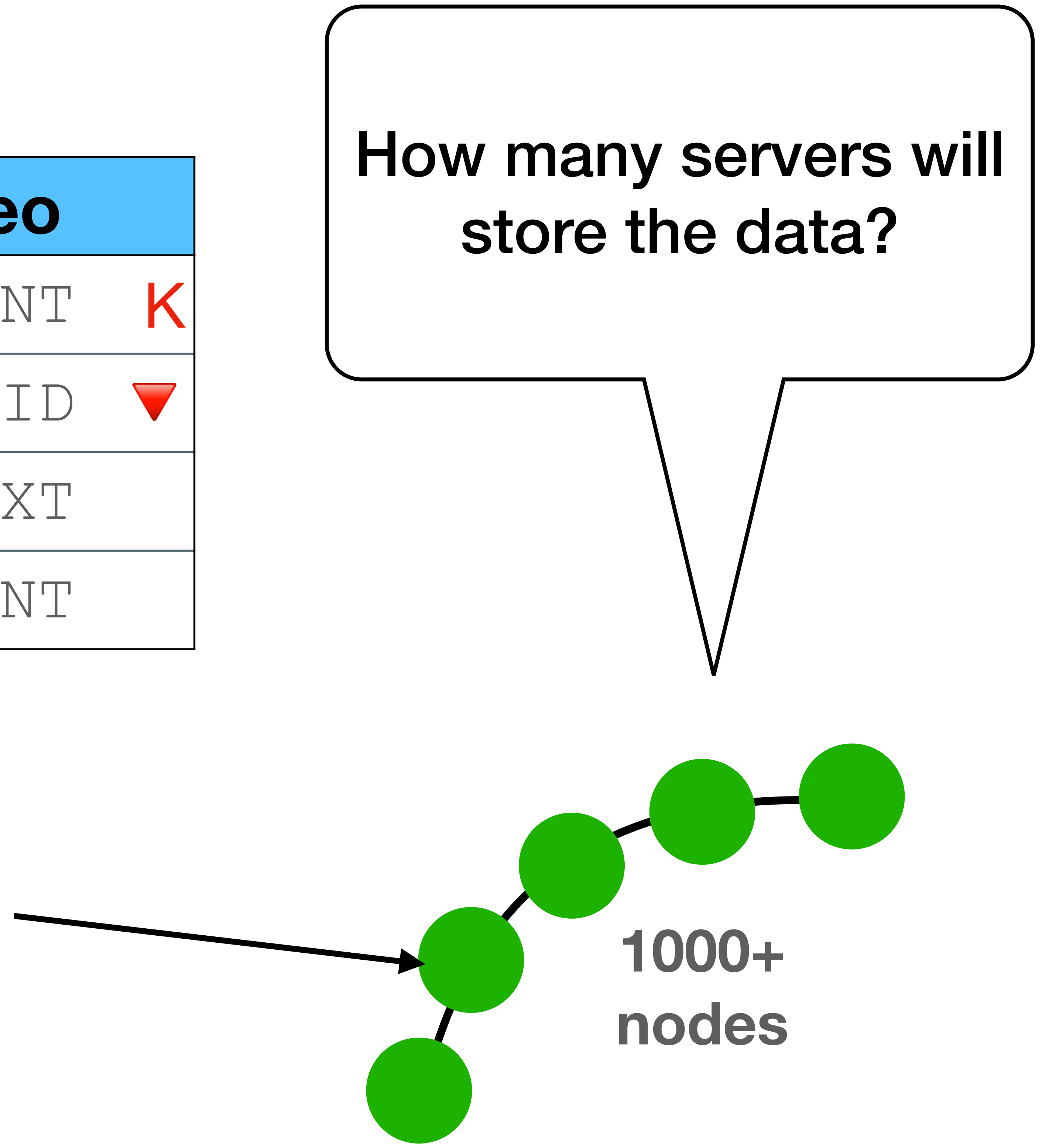
* images by Wikipedia

But what happens if the partition is “large”

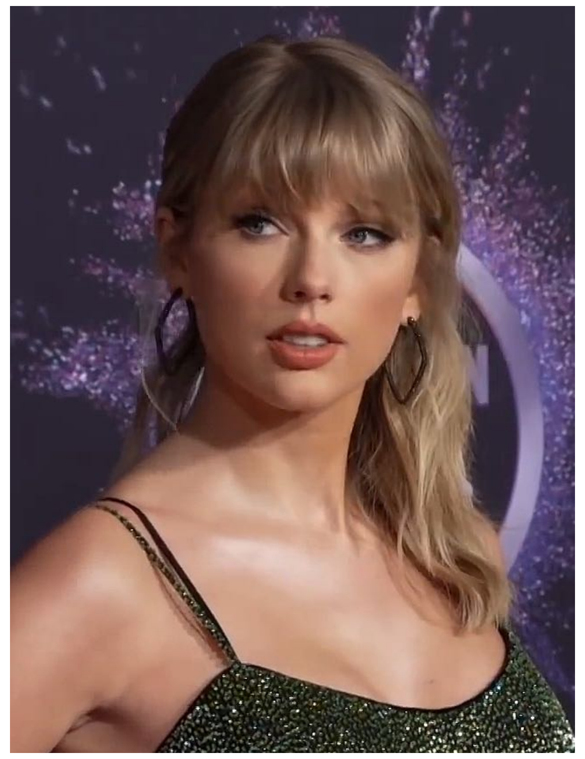
- There can be more than 10m rows in this partition

views_by_video

video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	



How many servers will store the data?



* images by Wikipedia

Large partitions

- Cause performance issues:
  - compactions are slower
  - queries are slower
  - repairs can fail
  - adding more nodes won’t help
- Can cause hotspots

  more on this later
- Data is not distributed evenly throughout the cluster
- We need to model differently to avoid

Large partitions in Cassandra

- Rule of thumb: partition size < 100MB size / 100k rows

  You can go higher with newer Cassandra versions

- You would need to estimate the size in advance

  Unless you learn the hard way you have a problem

How to avoid large partitions?

- What do you think?

How to avoid large partitions?

- The solution is easy:
  split the data into more partitions

- **When querying, the data is too big anyway for a single call**
  The driver automatically breaks the result into “pages” (default = 5000) even for a single partition

How to avoid large partitions?

- The solution is easy:
  split the data into more partitions

- **When querying, the data is too big anyway for a single call**
  The driver automatically breaks the result into “pages” (default = 5000) even for a single partition

How to split is the name of the game

“Choosing how to partition the data is not trivial,

it is hard.”

What is a good split?

views_by_user		
user_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
video_id	BIGINT	

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

What is a good split?

This is great as a single user probably won’t view over 100k videos

views_by_user		
user_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
video_id	BIGINT	

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

What is a good split?

This is great as a single user probably won't view over 100k videos

views_by_user		
user_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
video_id	BIGINT	

Problematic as some videos has more than 10m views

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

What is a good split?

This is great as a single user probably won't view over 100k videos

Might be problematic for a social network addict: assuming 100 views per day, we reach 100K after 1k days

views_by_user		
user_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
video_id	BIGINT	

Problematic as some videos has more than 10m views

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

What is a good split?

This is great as a single user probably won't view over 100k videos

Problematic as some videos has more than 10m views

views_by_user		
user_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
video_id	BIGINT	

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

Might be problematic for a social network addict: assuming 100 views per day, we reach 100K after 1k days

It depends on the query we need to answer

AND the data distribution

Points to remember when splitting

- Size limit
  large partitions causes performance issues
- Over shrinking
  when querying, it is better to contact 1 partition with 10k rows vs 10k partitions with 1 row
- “Known” partition keys
  when querying, the values of the partition keys are needed
- Hot spots
  undistributed writes/reads causes performance issues
- Tombstones
  too much deletes within a partition causes performance issues

Points to remember when splitting

- Size limit
  large partitions causes performance issues

Points to remember when splitting

- Size limit
  large partitions causes performance issues

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

10m views for a single video

Points to remember when splitting

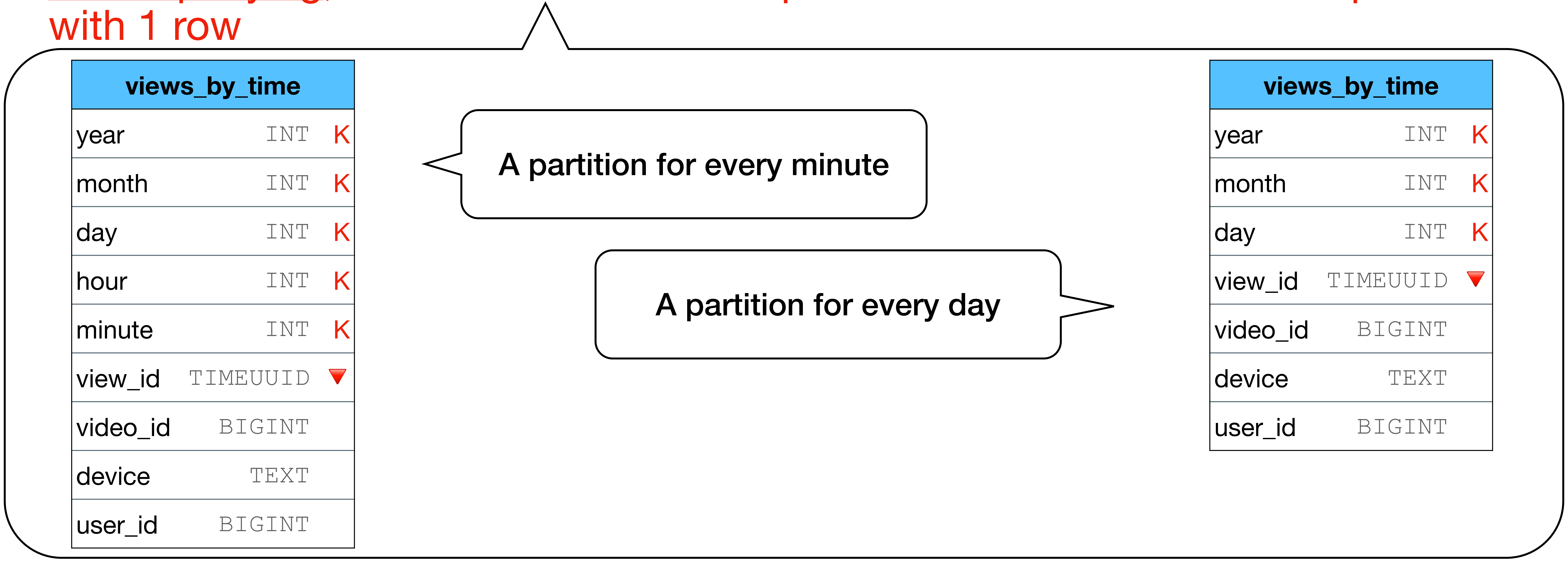
- Over shrinking

  when querying, it is better to contact 1 partition with 10k rows vs 10k partitions with 1 row

Points to remember when splitting

- Over shrinking

when querying, it is better to contact 1 partition with 10k rows vs 10k partitions with 1 row



views_by_time		
year	INT	K
month	INT	K
day	INT	K
hour	INT	K
minute	INT	K
view_id	TIMEUUID	▼
video_id	BIGINT	
device	TEXT	
user_id	BIGINT	

A partition for every minute

A partition for every day

views_by_time		
year	INT	K
month	INT	K
day	INT	K
view_id	TIMEUUID	▼
video_id	BIGINT	
device	TEXT	
user_id	BIGINT	

Points to remember when splitting

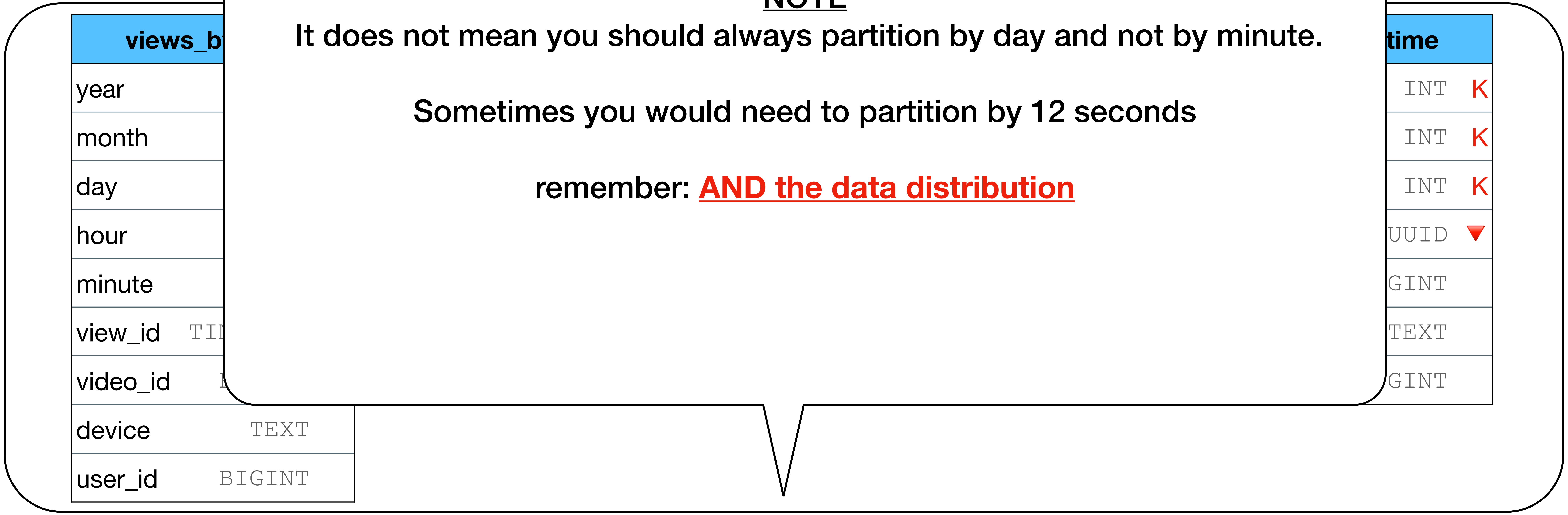
- Over sh
  when que artitions
  with 1 rov

NOTE

It does not mean you should always partition by day and not by minute.

Sometimes you would need to partition by 12 seconds

remember: **AND the data distribution**



views_b	
year	
month	
day	
hour	
minute	
view_id	TI
video_id	I
device	TEXT
user_id	BIGINT

time	
INT	K
INT	K
INT	K
UUID	▼
GINT	
TEXT	
GINT	

Points to remember when splitting

- “Known” partition keys
  when querying, the values of the partition keys are needed

Points to remember when splitting

- “Known” partition keys

  when querying, the values of the partition keys are needed

views_by_view		
view_id	TIMEUUID	K
video_id	BIGINT	
device	TEXT	
user_id	BIGINT	

How can we know the view_id values?

Points to remember when splitting

- Hot spots
  undistributed writes/reads causes performance issues

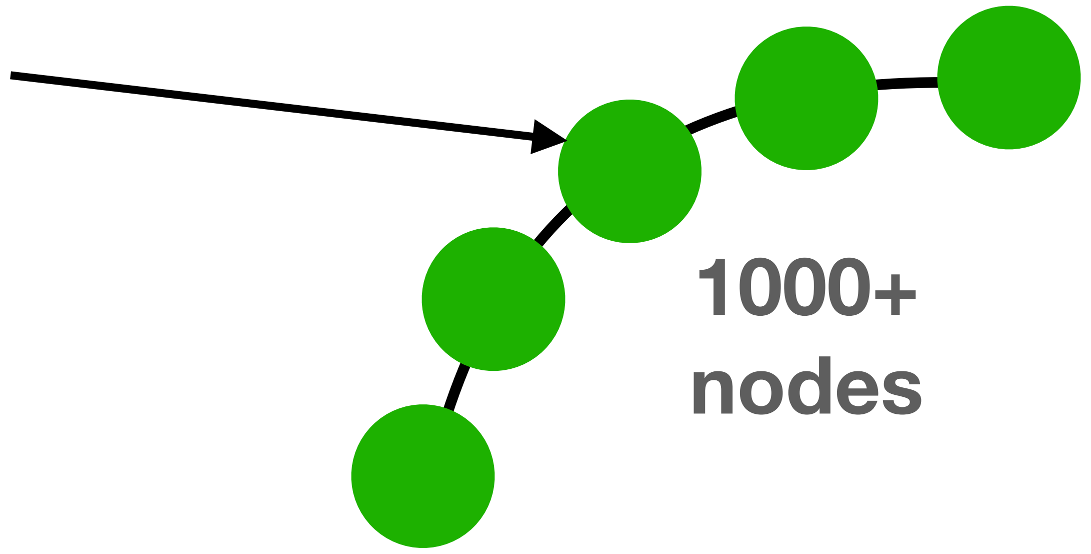
Points to remember when splitting

views_by_time		
year	INT	K
month	INT	K
day	INT	K
view_id	TIMEUUID	▼
video_id	BIGINT	
device	TEXT	
user_id	BIGINT	

- During each day only 1 node handles all the writes
- Assuming a 10k node cluster, 9999 server are unused (CPU & Storage)

- Hot spots

  undistributed writes/reads causes performance issues

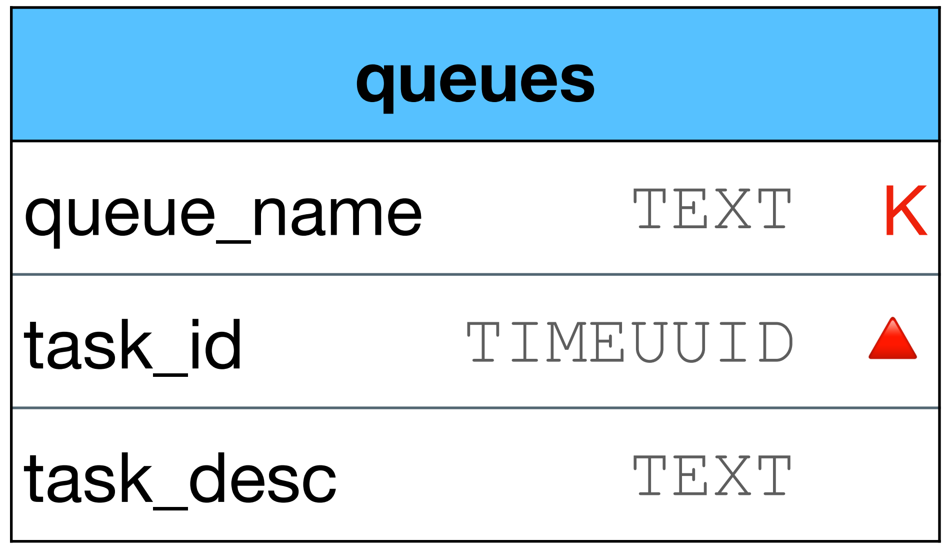


Points to remember when splitting

- Tombstones

  too much deletes within a partition causes performance issues

Points to remember when splitting



queues		
queue_name	TEXT	K
task_id	TIMEUUID	▲
task_desc	TEXT	

A queue for managing tasks (FIFO)
Once a task is done, it is deleted from the queue

Recall - during `gc_grace_seconds` (10 days):

- Warnings after 1k tombstones
- Partition crash after 100k tombstones

- Tombstones
  too much deletes within a partition causes performance issues

Again - this is important!

- Size limit
  large partitions causes performance issues
- Over shrinking
  when querying, it is better to contact 1 partition with 10k rows vs 10k partitions with 1 row
- “Known” partition keys
  when querying, the values of the partition keys are needed
- Hot spots
  undistributed writes/reads causes performance issues
- Tombstones
  too much deletes within a partition causes performance issues

Splitting strategies

- You can NOT satisfy all requirements for any strategy

- One is not better or worse than the other
  only more suitable to a specific example and data distribution

- Goal: learn different strategies and match the best model to each different problem

Option 1 - split with existing column

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

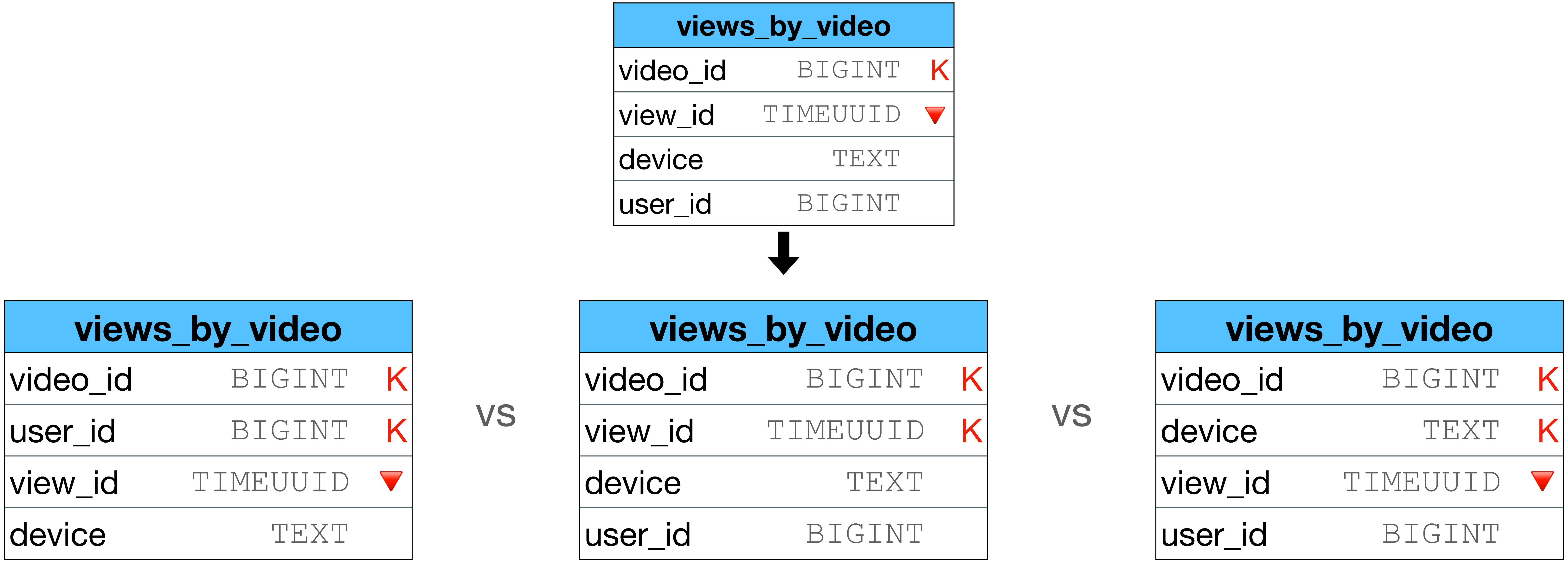
↓

Option 1 - split with existing column

Note - the query needed is “by video” although we add more partition keys

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

Option 1 - split with existing column



views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video		
video_id	BIGINT	K
user_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	

vs

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	K
device	TEXT	
user_id	BIGINT	

vs

views_by_video		
video_id	BIGINT	K
device	TEXT	K
view_id	TIMEUUID	▼
user_id	BIGINT	

Option 1 - split with existing column

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video		
video_id	BIGINT	K
user_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	

vs

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	K
device	TEXT	
user_id	BIGINT	

vs

views_by_video		
video_id	BIGINT	K
device	TEXT	K
view_id	TIMEUUID	▼
user_id	BIGINT	

size limit
over shrinking
known partitions
hot spots
tombstones

Option 1 - split with existing column

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

views_by_video		
video_id	BIGINT	K
user_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	

- 👍 size limit
- ❌ over shrinking
- ❌ known partitions
- 👍 hot spots
- 👍 tombstones

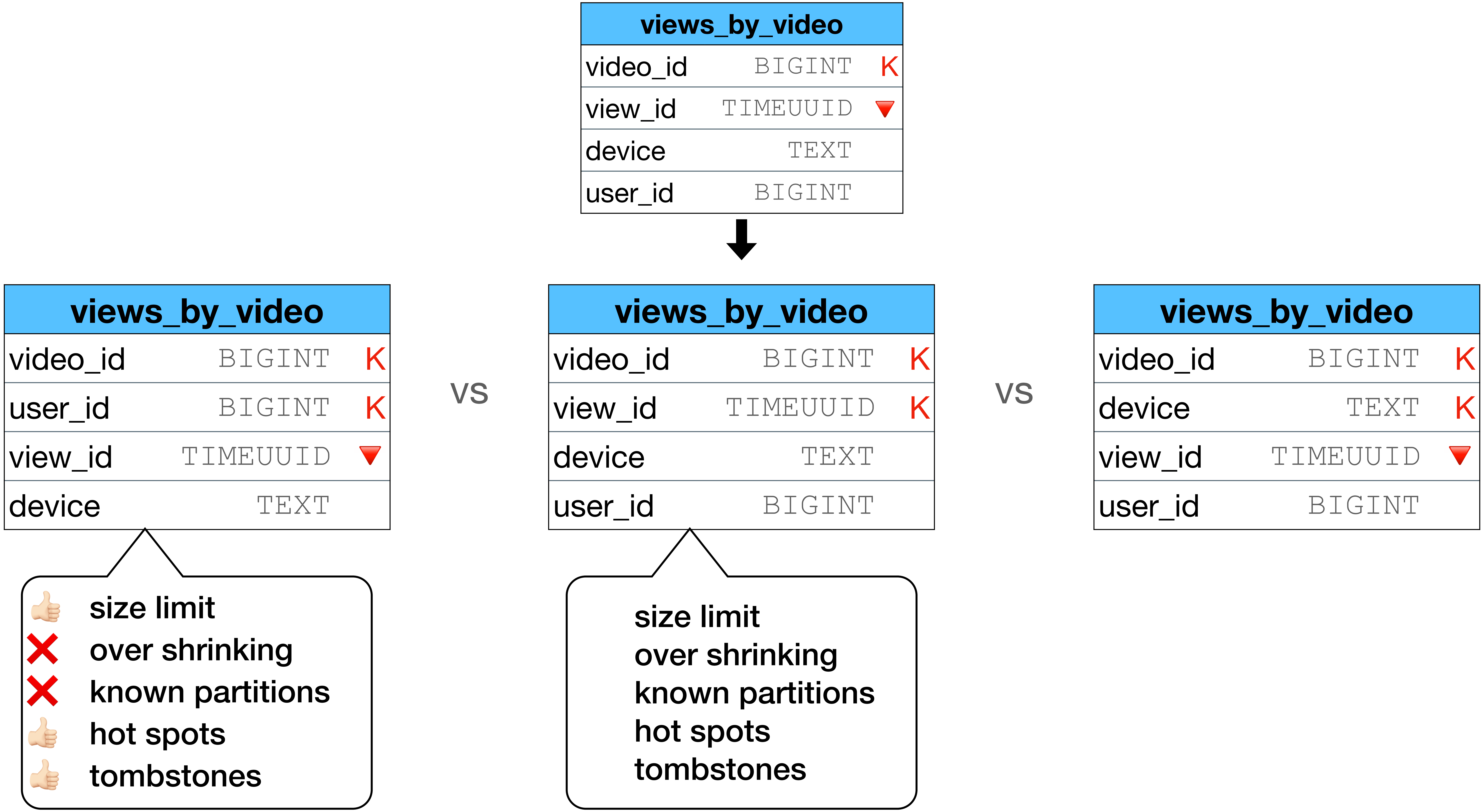
vs

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	K
device	TEXT	
user_id	BIGINT	

vs

views_by_video		
video_id	BIGINT	K
device	TEXT	K
view_id	TIMEUUID	▼
user_id	BIGINT	

Option 1 - split with existing column



views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

views_by_video		
video_id	BIGINT	K
user_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	

👍 size limit
❌ over shrinking
❌ known partitions
👍 hot spots
👍 tombstones

vs

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	K
device	TEXT	
user_id	BIGINT	

size limit
over shrinking
known partitions
hot spots
tombstones

vs

views_by_video		
video_id	BIGINT	K
device	TEXT	K
view_id	TIMEUUID	▼
user_id	BIGINT	

Option 1 - split with existing column

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

views_by_video		
video_id	BIGINT	K
user_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	

- 👍 size limit
- ❌ over shrinking
- ❌ known partitions
- 👍 hot spots
- 👍 tombstones

vs

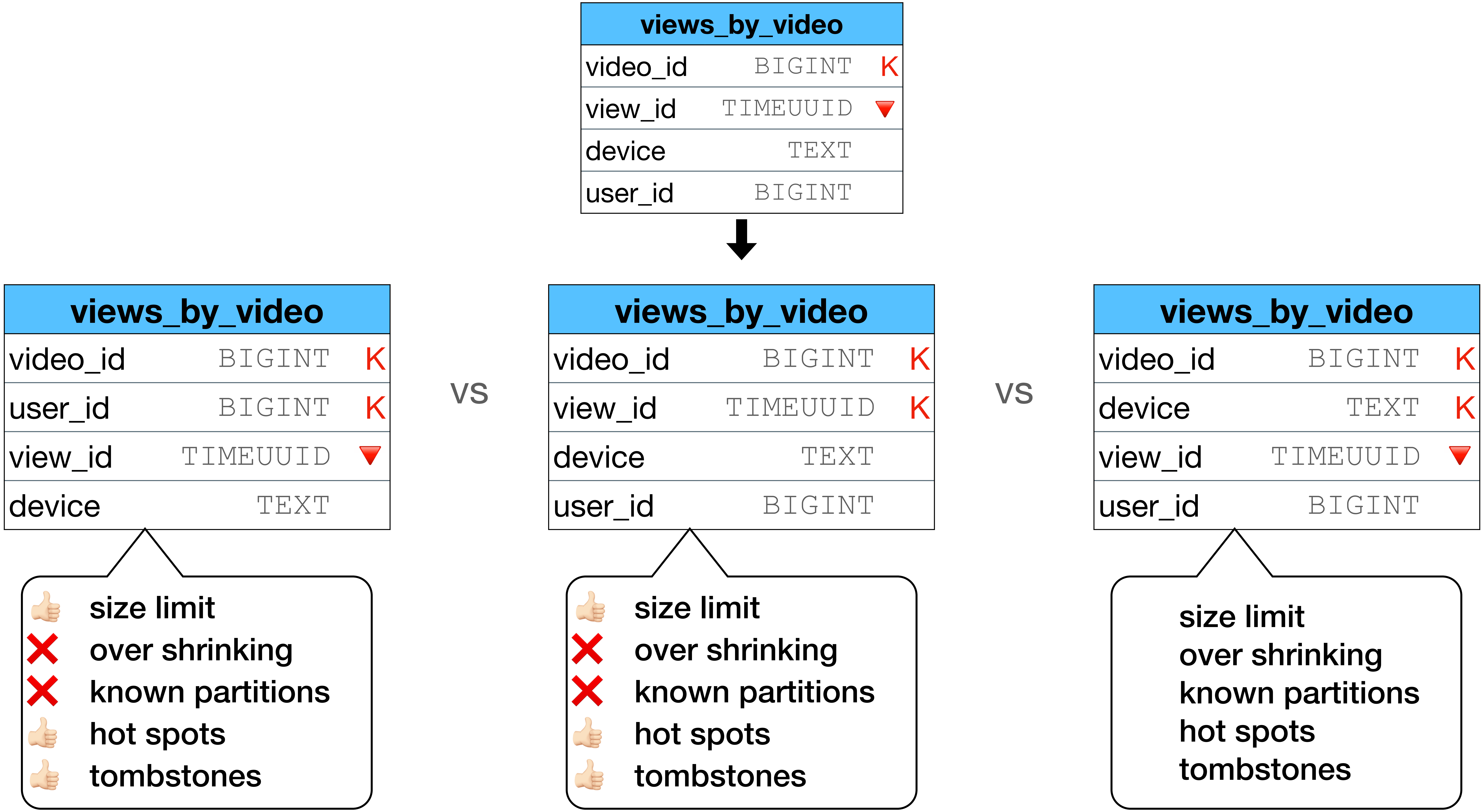
views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	K
device	TEXT	
user_id	BIGINT	

- 👍 size limit
- ❌ over shrinking
- ❌ known partitions
- 👍 hot spots
- 👍 tombstones

vs

views_by_video		
video_id	BIGINT	K
device	TEXT	K
view_id	TIMEUUID	▼
user_id	BIGINT	

Option 1 - split with existing column



views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

views_by_video		
video_id	BIGINT	K
user_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	

- 👍 size limit
- ❌ over shrinking
- ❌ known partitions
- 👍 hot spots
- 👍 tombstones

vs

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	K
device	TEXT	
user_id	BIGINT	

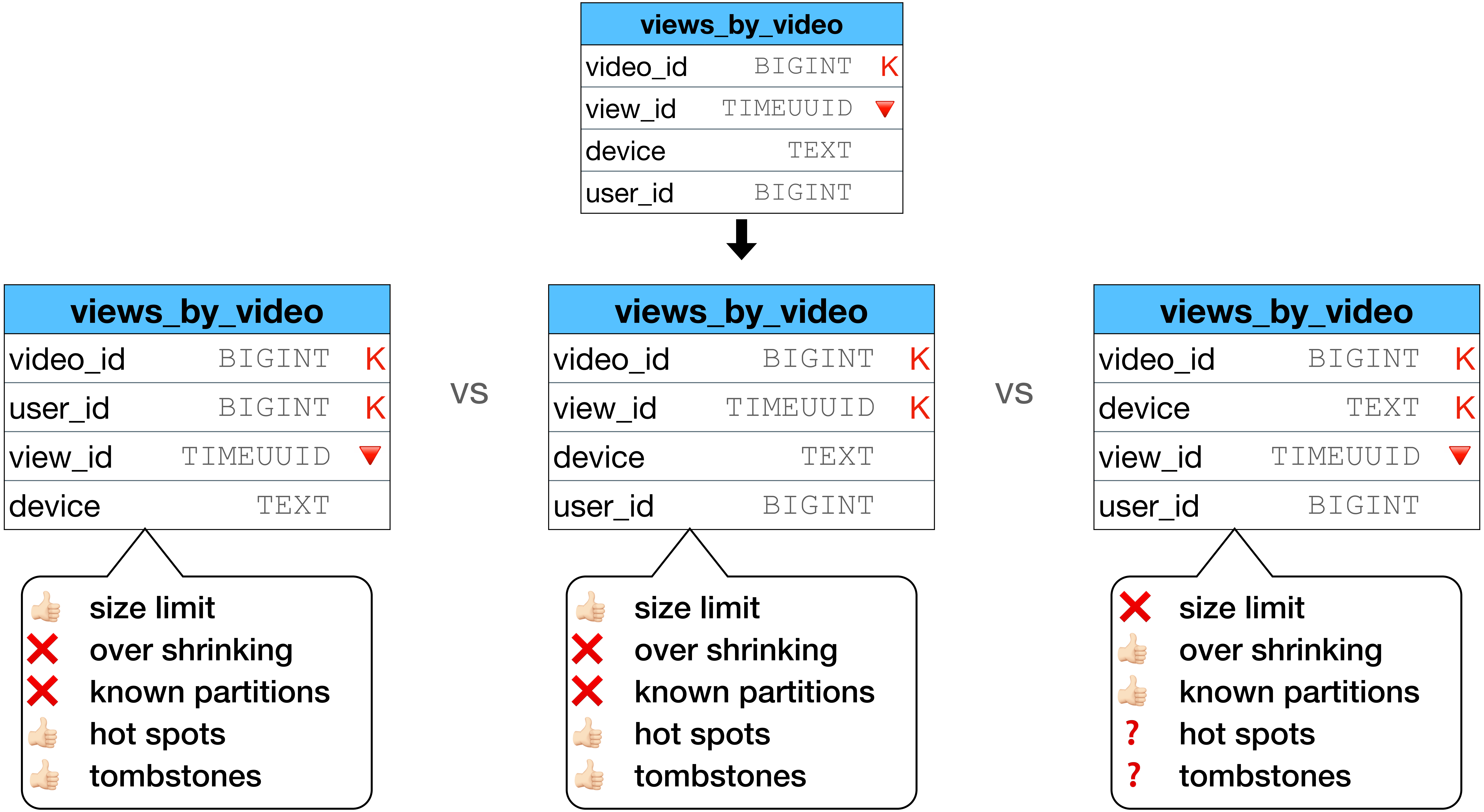
- 👍 size limit
- ❌ over shrinking
- ❌ known partitions
- 👍 hot spots
- 👍 tombstones

vs

views_by_video		
video_id	BIGINT	K
device	TEXT	K
view_id	TIMEUUID	▼
user_id	BIGINT	

size limit
over shrinking
known partitions
hot spots
tombstones

Option 1 - split with existing column



views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

views_by_video		
video_id	BIGINT	K
user_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	

- 👍 size limit
- ❌ over shrinking
- ❌ known partitions
- 👍 hot spots
- 👍 tombstones

vs

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	K
device	TEXT	
user_id	BIGINT	

- 👍 size limit
- ❌ over shrinking
- ❌ known partitions
- 👍 hot spots
- 👍 tombstones

vs

views_by_video		
video_id	BIGINT	K
device	TEXT	K
view_id	TIMEUUID	▼
user_id	BIGINT	

- ❌ size limit
- 👍 over shrinking
- 👍 known partitions
- ? hot spots
- ? tombstones

Option 2 - split with artificial (time) column

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

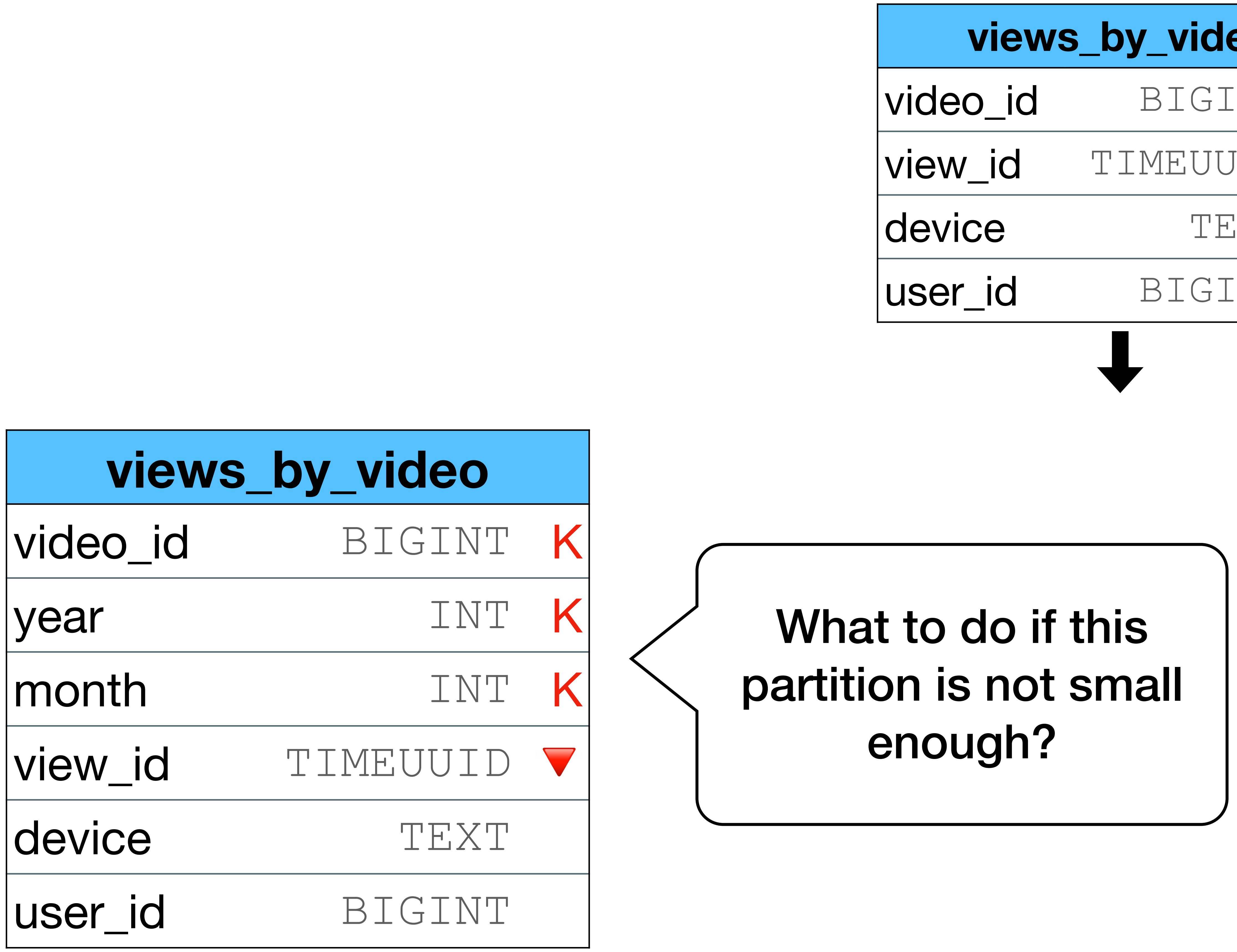
Option 2 - split with artificial (time) column

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

↓

views_by_video		
video_id	BIGINT	K
year	INT	K
month	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

Option 2 - split with artificial (time) column

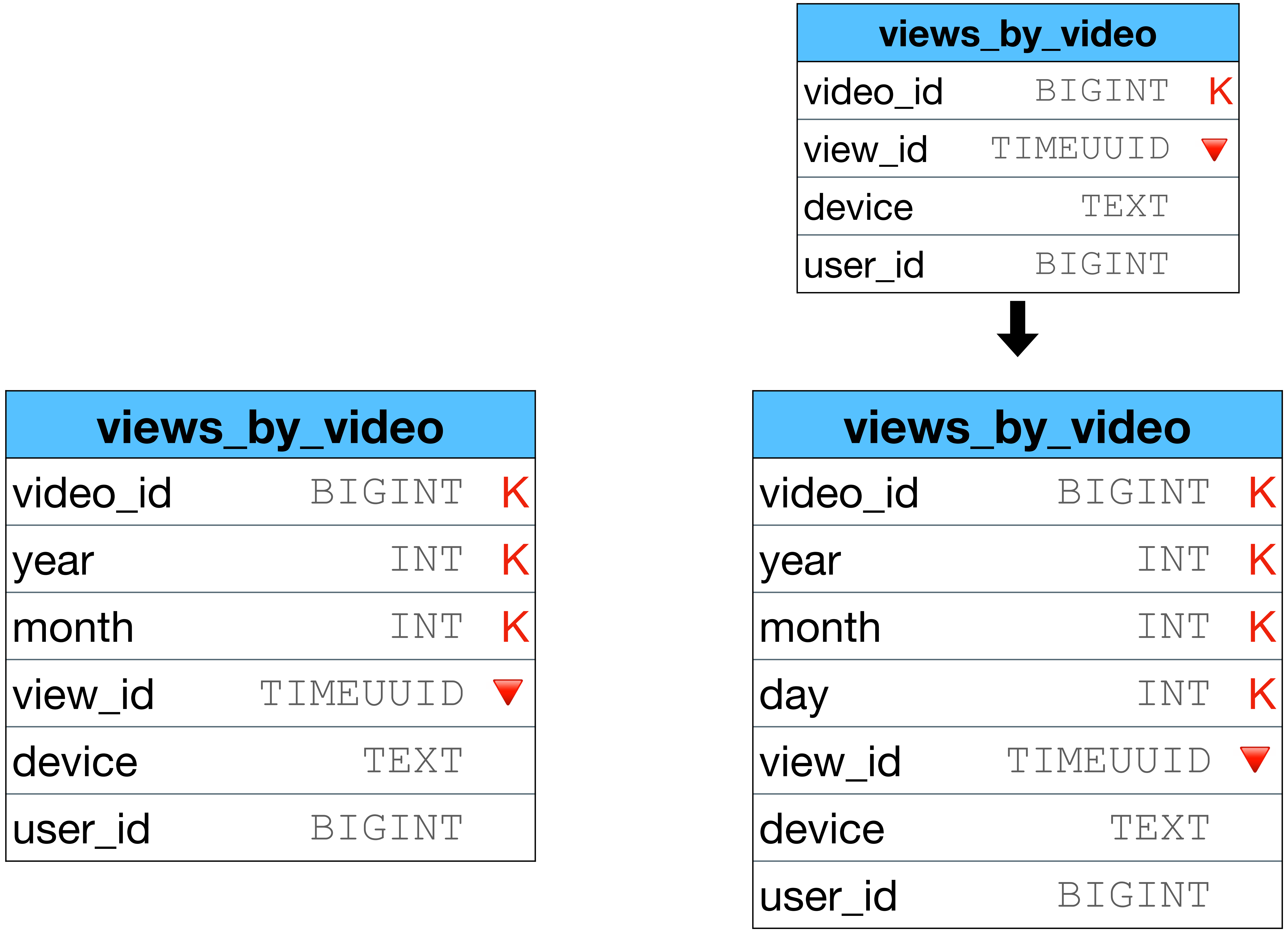


views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video		
video_id	BIGINT	K
year	INT	K
month	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

What to do if this partition is not small enough?

Option 2 - split with artificial (time) column

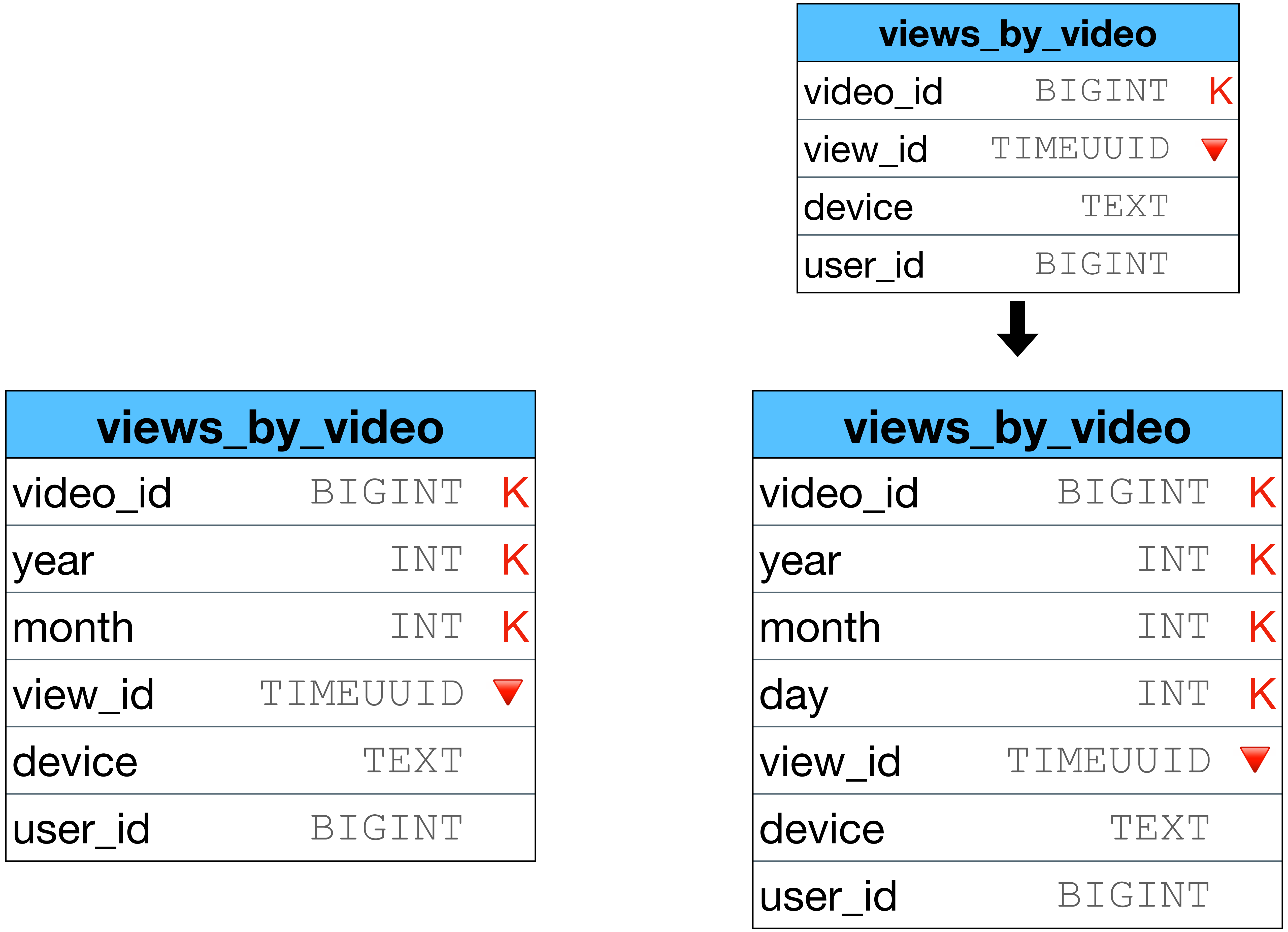


views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video		
video_id	BIGINT	K
year	INT	K
month	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video		
video_id	BIGINT	K
year	INT	K
month	INT	K
day	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

Option 2 - split with artificial (time) column



views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

views_by_video		
video_id	BIGINT	K
year	INT	K
month	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video		
video_id	BIGINT	K
year	INT	K
month	INT	K
day	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

We can have the same problem. How can we solve it <u>without the need to change the schema each time</u>?

Option 2 - split with artifi

Assume the time is 2021/12/22 14:54:34:3233

Round the TS before you insert the data

- By year use 2021/01/01 00:00:00:0000
- By month use 2021/12/01 00:00:00:0000
- By day use 2021/12/22 00:00:00:0000
- By hour use 2021/12/22 14:00:00:0000
- By minute use 2021/12/22 14:54:00:0000
- …
- * use GMT=0 to avoid timezones / daylight

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

views_by_video		
video_id	BIGINT	K
year	INT	K
month	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video		
video_id	BIGINT	K
year	INT	K
month	INT	K
day	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video		
video_id	BIGINT	K
ts_partition	TIMESTAMP	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

Option 2 - split with artificial (time) column

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

views_by_video		
video_id	BIGINT	K
year	INT	K
month	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video		
video_id	BIGINT	K
year	INT	K
month	INT	K
day	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video		
video_id	BIGINT	K
ts_partition	TIMESTAMP	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

size limit
over shrinking
known partitions
hot spots
tombstones

Option 2 - split with artificial (time) column

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

views_by_video		
video_id	BIGINT	K
year	INT	K
month	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video		
video_id	BIGINT	K
year	INT	K
month	INT	K
day	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

For most days ok, except aired date of new episodes

views_by_video		
video_id	BIGINT	K
ts_partition	TIMESTAMP	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

? size limit
? over shrinking
👍 known partitions
? hot spots
? tombstones

Option 2 - split with artificial (time) column

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

views_by_video		
video_id	BIGINT	K
year	INT	K
month	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video		
video_id	BIGINT	K
year	INT	K
month	INT	K
day	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video		
video_id	BIGINT	K
ts_partition	TIMESTAMP	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

Note - “by minute” might be needed for “Game of Thrones” but not for all other 5000 shows

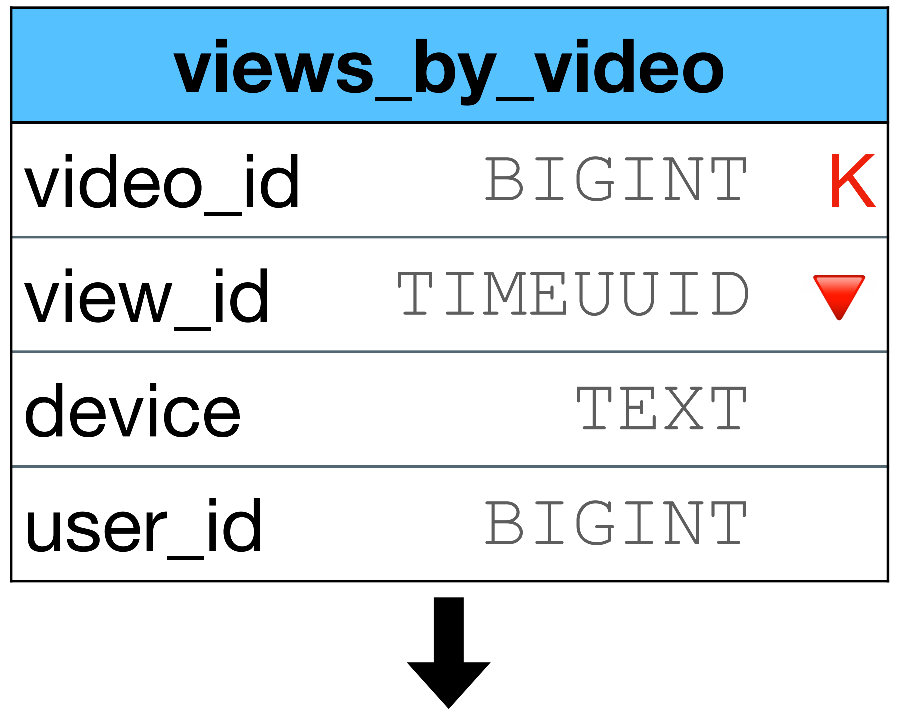
For most days ok, except aired date of new episodes

- ? size limit
- ? over shrinking
- 👍 known partitions
- ? hot spots
- ? tombstones

Option 3 - split with bucket column

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

Option 3 - split with bucket column



views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

- Start with bucket 0.

- If more than X (50k?) views, advance to bucket 1

- …

Option 3 - split with bucket column

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

views_by_video		
video_id	BIGINT	K
bucket	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

- Start with bucket 0.
- If more than X (50k?) views, advance to bucket 1
- …

views_by_video_buckets		
video_id	BIGINT	K
buckets	INT	▼
views	COUNTER	++

This table will help us "count" the number of view per bucket

Option 3 - split with bucket column

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

↓

- Start with bucket 0.
- If more than X (50k?) views, advance to bucket 1
- …

views_by_video		
video_id	BIGINT	K
bucket	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

size limit
over shrinking
known partitions
hot spots
tombstones

This table will help us “count” the number of view per bucket

views_by_video_buckets		
video_id	BIGINT	K
buckets	INT	▼
views	COUNTER	++

Option 3 - split with bucket column

views_by_video

video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

- Start with bucket 0.

- If more than X (50k?) views, advance to bucket 1

- …

views_by_video

video_id	BIGINT	K
bucket	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

👍 size limit
👍 over shrinking
👍 known partitions
? hot spots
👍 tombstones

Great option, but not trivial to maintain the logic on the backend

This table will help us "count" the number of view per bucket

views_by_video_buckets

video_id	BIGINT	K
buckets	INT	▼
views	COUNTER	++

bucket column

Pros

- Guaranteed max size

- Can grow without a limit

- When queuing - optimized for the number of calls
  - we do not have “small” partitions

- Ordered by TS across all partitions (<u>only if we always add “new” data</u>)

Cons

- If we add “old” data, the TS is NOT ordered across all partitions

- We can NOT “find” a specific event as we do not know on which partition the data is saved
  in the example - we can NOT know if a specific view_id exists without reading all partitions

deo		
	GINT	K
	UUID	▼
	TEXT	
	GINT	

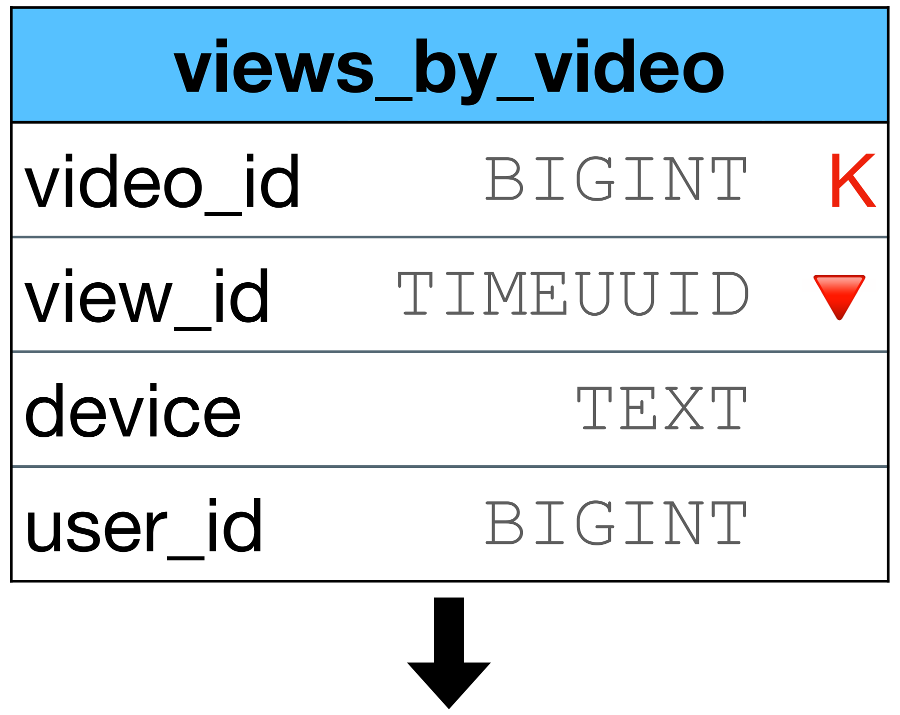
ideo		
	IGINT	K
	INT	K
	EUUID	▼
	TEXT	
	IGINT	

_buckets		
	BIGINT	K
buckets	INT	▼
views	COUNTER	++

👍 size limit
👍 over shrinking
👍 known partitions
? hot spots
👍 tombstones

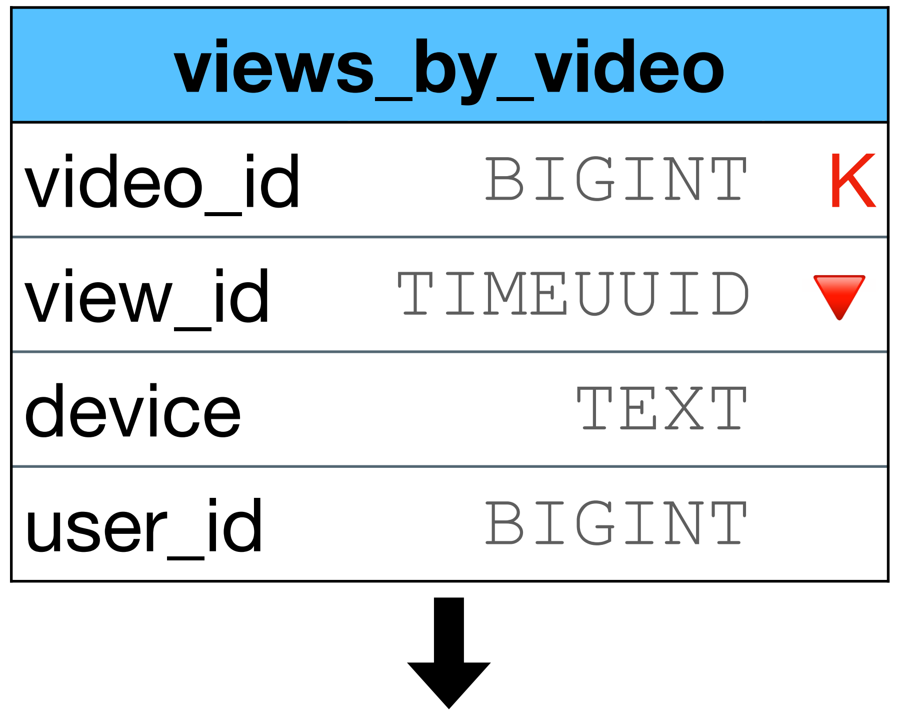
Great option,
but not trivial to maintain the
logic on the backend

Option 4 - split with partition column



views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

Option 4 - split with partition column



views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

- Decide on max partition size (1000?)

- Use a “hash function” to distribute the data evenly across the partition

Option 4 - split with partition column

views_by_video

video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

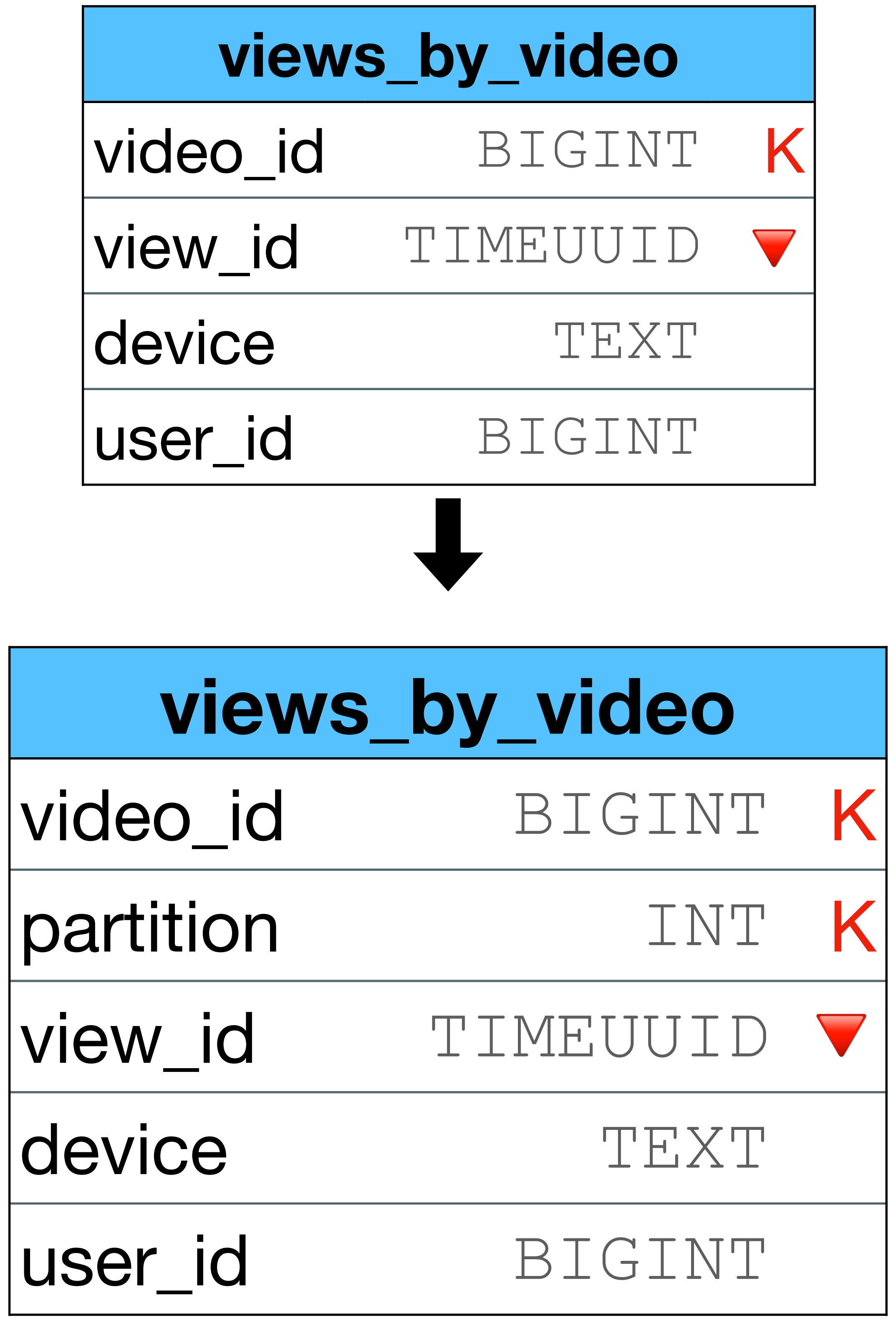
⬇

views_by_video

video_id	BIGINT	K
partition	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

- Decide on max partition size (1000?)
- Use a “hash function” to distribute the data evenly across the partition

Option 4 - split with partition column



views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

views_by_video		
video_id	BIGINT	K
partition	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

- Decide on max partition size (1000?)

- Use a “hash function” to distribute the data evenly across the partition

- For example modulo:
  `partition = user_id % 1000`

Option 4 - split with partition column

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

↓

views_by_video		
video_id	BIGINT	K
partition	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

- Decide on max partition size (1000?)
- Use a “hash function” to distribute the data evenly across the partition
- For example modulo:
  `partition = user_id % 1000`

Data is distributed evenly

Option 4 - split with partition column

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

views_by_video		
video_id	BIGINT	K
partition	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

size limit
over shrinking
known partitions
hot spots
tombstones

- Decide on max partition size (1000?)
- Use a “hash function” to distribute the data evenly across the partition
- For example modulo:

```
partition =
user_id % 1000
```

Option 4 - split with partition column

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

views_by_video		
video_id	BIGINT	K
partition	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

- Decide on max partition size (1000?)
- Use a “hash function” to distribute the data evenly across the partition
- For example modulo:
  `partition = user_id % 1000`

👍 size limit
❓ over shrinking
👍 known partitions
👍 hot spots
👍 tombstones

Not all videos need the same partition size

Option 4 - split with partition column

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

views_by_video		
video_id	BIGINT	K
partition	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

- Decide on max partition size (1000?)

- Use a “hash function” to distribute the data evenly across the partition

- For example modulo:
`partition = user_id % 1000`

What about the order of the data?

- 👍 size limit
- ? over shrinking
- 👍 known partitions
- 👍 hot spots
- 👍 tombstones

Not all videos need the same partition size

Option 4 - split with partition column

- Decide on max partition size (1000?)

- Use a “hash function” to distribute the data evenly across the partition

- For example modulo:
```
partition =
user_id % 1000
```

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

views_by_video		
video_id	BIGINT	K
partition	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

When we read the data, it is NOT ordered by the “global” view_id, but per partition.

Can (maybe) cause logic problems for the client

👍	size limit
?	over shrinking
👍	known partitions
👍	hot spots
👍	tombstones

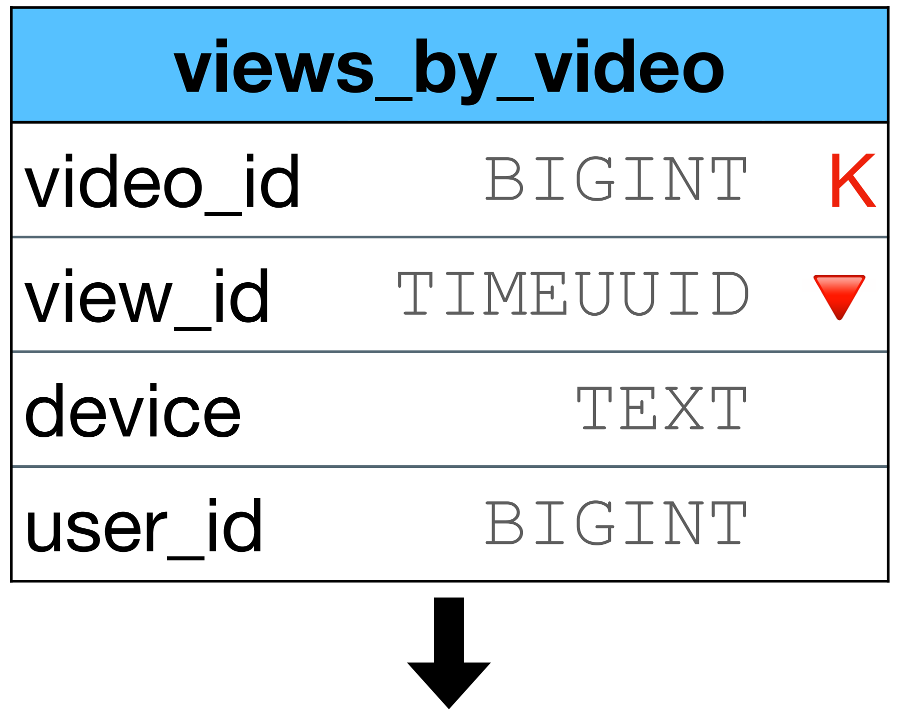
Not all videos need the same partition size

Option 5 - combo (variable partition size)

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

Option 5 - combo (variable partition size)



views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

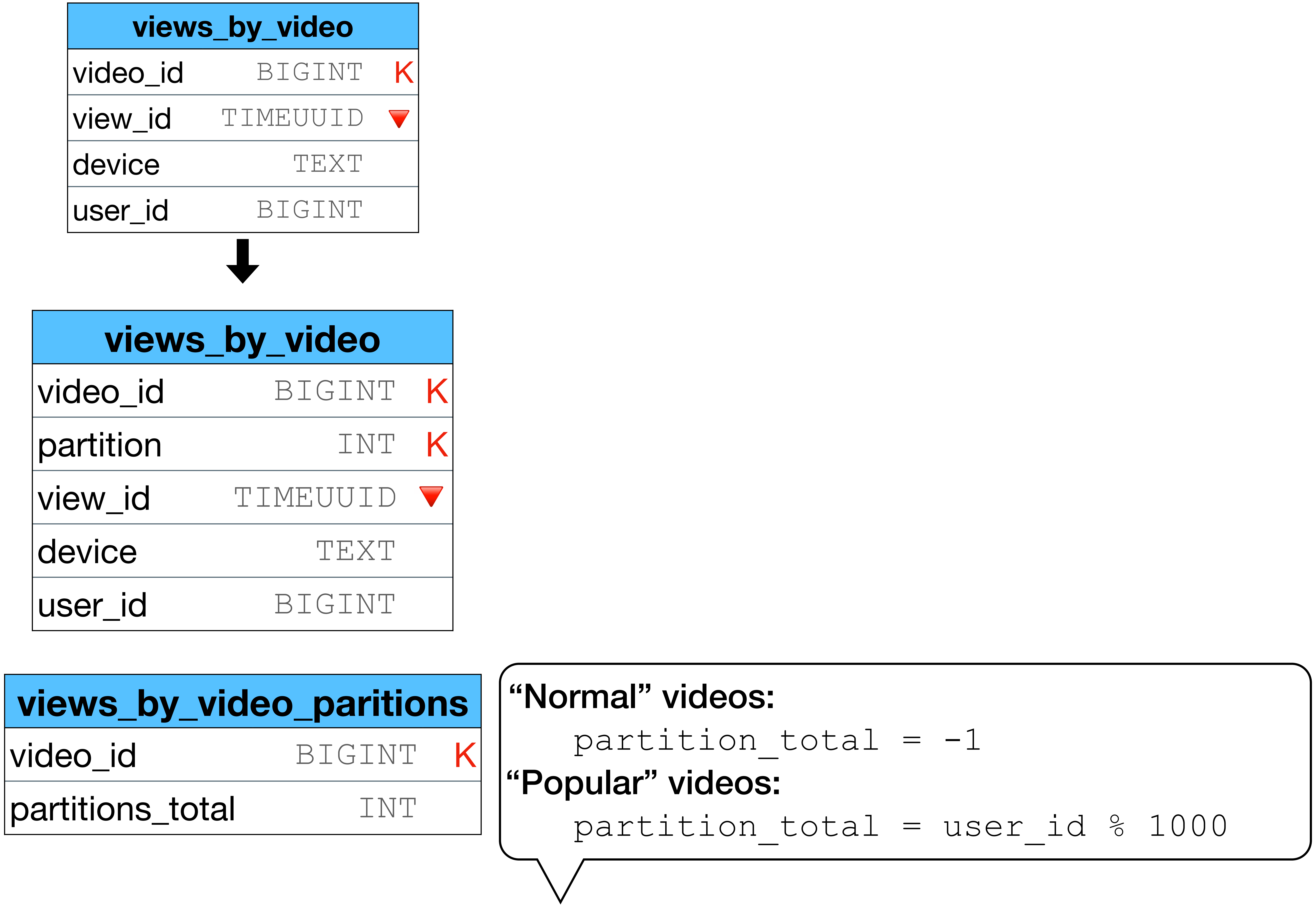
- Variable max partition size per video

- Use a “hash function” to distribute the data evenly across the partition (<u>with special logic</u>)

Option 5 - combo (variable partition size)

- Variable max partition size per video

- Use a “hash function” to distribute the data evenly across the partition (with special logic)



views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video		
video_id	BIGINT	K
partition	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video_paritions		
video_id	BIGINT	K
partitions_total	INT	

“Normal” videos:

```
partition_total = -1
```

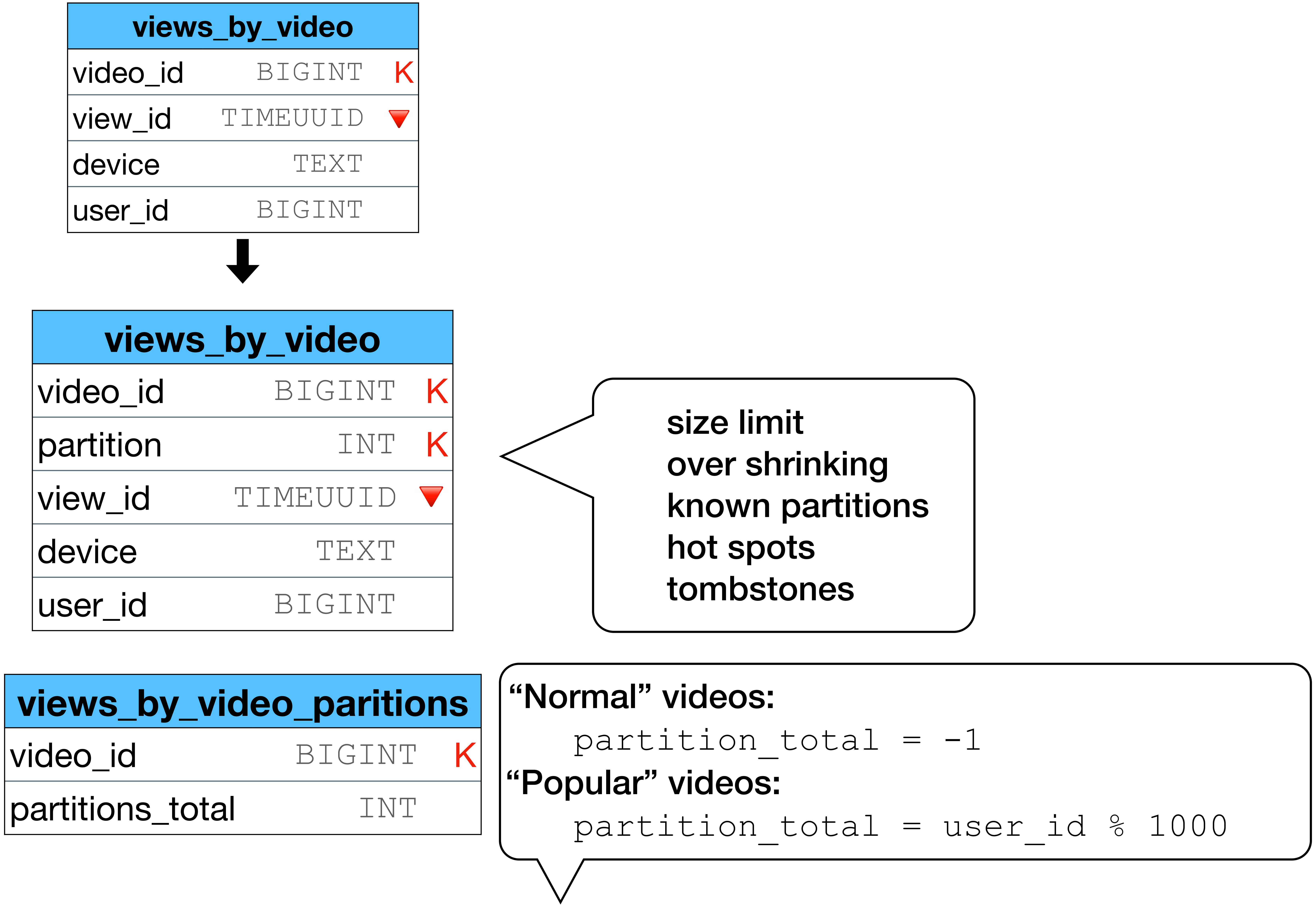
“Popular” videos:

```
partition_total = user_id % 1000
```

Option 5 - combo (variable partition size)

- Variable max partition size per video

- Use a “hash function” to distribute the data evenly across the partition (with special logic)



views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video		
video_id	BIGINT	K
partition	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

size limit
over shrinking
known partitions
hot spots
tombstones

views_by_video_paritions		
video_id	BIGINT	K
partitions_total	INT	

“Normal” videos:

```
    partition_total = -1
```

“Popular” videos:

```
    partition_total = user_id % 1000
```

Option 5 - combo (variable partition size)

- Variable max partition size per video

- Use a “hash function” to distribute the data evenly across the partition (with special logic)

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

↓

views_by_video		
video_id	BIGINT	K
partition	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video_paritions		
video_id	BIGINT	K
partitions_total	INT	

A logic is required to set the right partitions_total for each video

👍 size limit
👍 over shrinking
👍 known partitions
👍 hot spots
👍 tombstones

“Normal” videos:

```
partition_total = -1
```

“Popular” videos:

```
partition_total = user_id % 1000
```

Option 5 - combo (variable partition size)

- Variable max partition size per video

- Use a “hash function” to distribute the data evenly across the partition (with special logic)

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

⬇

views_by_video		
video_id	BIGINT	K
partition	INT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

views_by_video_paritions		
video_id	BIGINT	K
partitions_total	INT	

Discussion - why did we chose “-1” for “normal” users and not “0”

A logic is required to set the right partitions_total for each video

- 👍 size limit
- 👍 over shrinking
- 👍 known partitions
- 👍 hot spots
- 👍 tombstones

“Normal” videos:

```
    partition_total = -1
```

“Popular” videos:

```
    partition_total = user_id % 1000
```

riable partition size)

We want to support the option to “transition” state from “normal” to “poplar”

—> we need to use “different” partitions for each state in order to “reinsert” the data on “transition”

“Normal” videos:

```
    partition_total = -1
```

“Popular” videos:

```
    partition_total = user_id % 1000
```

“Super popular” videos:

```
    partition_total = 10000 + (user_id % 10000)
```

ID	▼
EXT	
INT	

deo	
GINT	K
INT	K
UUID	▼
TEXT	
GINT	

- distribute the data evenly across the partition (with special logic)

views_by_video_paritions		
video_id	BIGINT	K
partitions_total	INT	

Discussion - why did we chose “-1” for “normal” users and not “0”

A logic is required to set the right partitions_total for each video

- 👍 size limit
- 👍 over shrinking
- 👍 known partitions
- 👍 hot spots
- 👍 tombstones

“Normal” videos:

```
    partition_total = -1
```

“Popular” videos:

```
    partition_total = user_id % 1000
```

riable partition size)

We want to support the option to "transition" state from "normal" to "poplar"

—> we need to use "different" partitions for each state in order to "reinsert" the data on "transition"

"Normal" videos:

```
    partition_total = -1
```

"Popular" videos:

```
    partition_total = user_id % 1000
```

"Super popular" videos:

```
    partition_total = 10000 + (user_id % 10000)
```

Discussion - why did we chose "-1" for "normal" users and not "0"

ID
EXT
INT

deo

GINT	K
INT	K
UUID	▼
TEXT	
GINT	

A logic is required to set the right partitions_total for each video

- 👍 size limit
- 👍 over shrinking
- 👍 known partitions
- 👍 hot spots
- 👍 tombstones

distribute the data evenly across the partition (with special logic)

views_by_video_paritions

video_id	BIGINT	K
partitions_total	INT	

"Normal" videos:

```
    partition_total = -1
```

"Popular" videos:

```
    partition_total = user_id % 1000
```

Why did Instagram crushed?

- Instagram has different write paths for “top users”
  that is, different **data models** and different **app logic**
- There is an application logic that transition a user from a “regular” user to a “top user”
- **The (regular) data model used did not scaled**

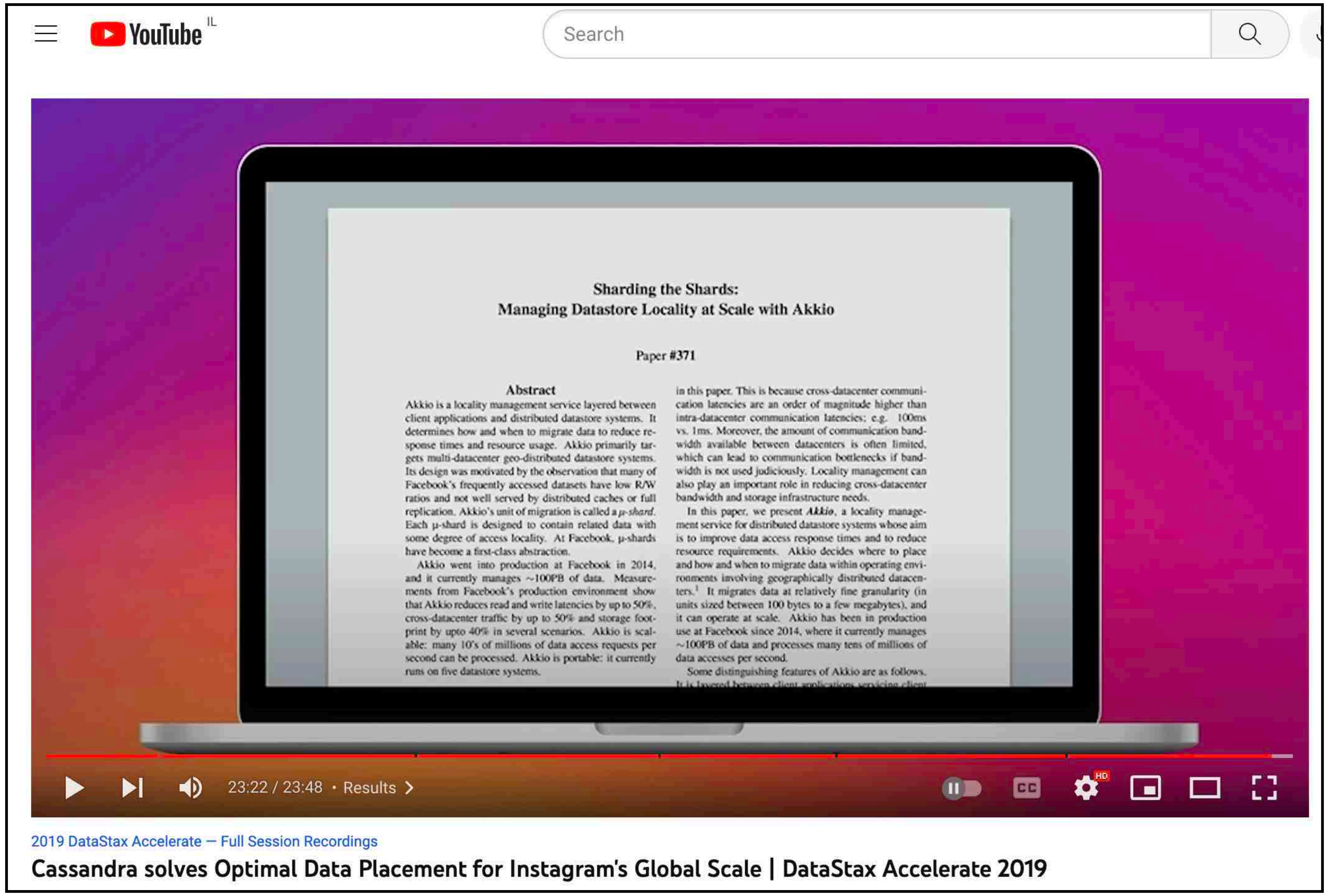
*1 - speculation

*2 - more info on “data modeling examples”

Splitting strategies - reminder

- One is not better or worse than the other
  only more suitable to a specific example and data distribution

When sharding is not enough…



Sharding the Shards:
Managing Datastore Locality at Scale with Akkio

Paper #371

Abstract

Akkio is a locality management service layered between client applications and distributed datastore systems. It determines how and when to migrate data to reduce response times and resource usage. Akkio primarily targets multi-datacenter geo-distributed datastore systems. Its design was motivated by the observation that many of Facebook's frequently accessed datasets have low R/W ratios and not well served by distributed caches or full replication. Akkio's unit of migration is called a *μ-shard*. Each μ-shard is designed to contain related data with some degree of access locality. At Facebook, μ-shards have become a first-class abstraction.

Akkio went into production at Facebook in 2014, and it currently manages ~100PB of data. Measurements from Facebook's production environment show that Akkio reduces read and write latencies by up to 50%, cross-datacenter traffic by up to 50% and storage footprint by upto 40% in several scenarios. Akkio is scalable: many 10's of millions of data access requests per second can be processed. Akkio is portable: it currently runs on five datastore systems.

in this paper. This is because cross-datacenter communication latencies are an order of magnitude higher than intra-datacenter communication latencies; e.g. 100ms vs. 1ms. Moreover, the amount of communication bandwidth available between datacenters is often limited, which can lead to communication bottlenecks if bandwidth is not used judiciously. Locality management can also play an important role in reducing cross-datacenter bandwidth and storage infrastructure needs.

In this paper, we present ***Akkio***, a locality management service for distributed datastore systems whose aim is to improve data access response times and to reduce resource requirements. Akkio decides where to place and how and when to migrate data within operating environments involving geographically distributed datacenters.[1] It migrates data at relatively fine granularity (in units sized between 100 bytes to a few megabytes), and it can operate at scale. Akkio has been in production use at Facebook since 2014, where it currently manages ~100PB of data and processes many tens of millions of data accesses per second.

Some distinguishing features of Akkio are as follows. It is layered between client applications servicing client

2019 DataStax Accelerate — Full Session Recordings

Cassandra solves Optimal Data Placement for Instagram's Global Scale | DataStax Accelerate 2019

Only if you are a
“data nerd”..

https://www.youtube.com/watch?v=Sr0sX-TId-g