Data Modeling in NoSQL (C*) - Intro

Big Data Systems

Dr. Rubi Boim

Data modeling - the most important property for big data systems

TLDR (1)

quick discussion - what does this means?
(example on next slides)

- Query-driven modeling
  (model for performance - goal: minimize partition reads)
  - Sacrifice space for (query) time
  - Denormalization - we materialize a JOIN on write vs on read

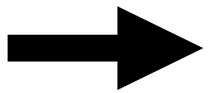
- “Forget” RDBMS
  - No JOINS
  - No referential integrity

```
CREATE TABLE users_by_id (
   user_id      BIGINT,
   fname        TEXT,
   lname        TEXT,
   country      TEXT,
   PRIMARY KEY (user_id)
);

CREATE TABLE users_by_country (
   country      TEXT,
   user_id      BIGINT,
   PRIMARY KEY (country, user_id)
);
```

How can we get all the data for all the users in Israel?

```
CREATE TABLE users_by_id (
   user_id      BIGINT,
   fname        TEXT,
   lname        TEXT,
   country      TEXT,
   PRIMARY KEY (user_id)
);

CREATE TABLE users_by_country (
   country      TEXT,
   user_id      BIGINT,
   PRIMARY KEY (country, user_id)
);
```

```
SELECT user_id
FROM users_by_country
WHERE country = 'Israel'
```



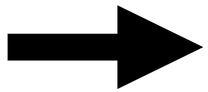
```
for (user:result) {
   SELECT * FROM users_by_id
   WHERE user_id = user
}
```

```
CREATE TABLE users_by_id (
   user_id      BIGINT,
   fname        TEXT,
   lname        TEXT,
   country      TEXT,
   PRIMARY KEY (user_id)
);
```

```
CREATE TABLE users_by_country (
   country      TEXT,
   user_id      BIGINT,
   PRIMARY KEY (country, user_id)
);
```

How many queries do we need?

```
SELECT user_id
FROM users_by_country
WHERE country = 'Israel'
```



```
for (user:result) {
   SELECT * FROM users_by_id
   WHERE user_id = user
}
```

```
CREATE TABLE users_by_id (
   user_id      BIGINT,
   fname        TEXT,
   lname        TEXT,
   country      TEXT,
   PRIMARY KEY (user_id)
);

CREATE TABLE users_by_country (
   country      TEXT,
   user_id      BIGINT,
   PRIMARY KEY (country, user_id)
);
```

```
CREATE TABLE users_by_id (
   user_id      BIGINT,
   fname        TEXT,
   lname        TEXT,
   country      TEXT,
   PRIMARY KEY (user_id)
);

CREATE TABLE users_by_country (
   country      TEXT,
   user_id      BIGINT,
   fname        TEXT,
   lname        TEXT,
   PRIMARY KEY (country, user_id)
);
```

```
CREATE TABLE users_by_id (
   user_id      BIGINT,
   fname        TEXT,
   lname        TEXT,
   country      TEXT,
   PRIMARY KEY (user_id)
);
```

```
CREATE TABLE users_by_country (
   country      TEXT,
   user_id      BIGINT,
   PRIMARY KEY (country, user_id)
);
```


```
CREATE TABLE users_by_id (
   user_id      BIGINT,
   fname        TEXT,
   lname        TEXT,
   country      TEXT,
   PRIMARY KEY (user_id)
);
```

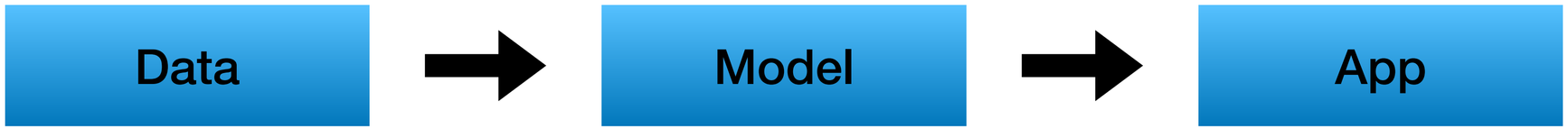
```
CREATE TABLE users_by_country (
   country      TEXT,
   user_id      BIGINT,
   fname        TEXT,
   lname        TEXT,
   PRIMARY KEY (country, user_id)
);
```



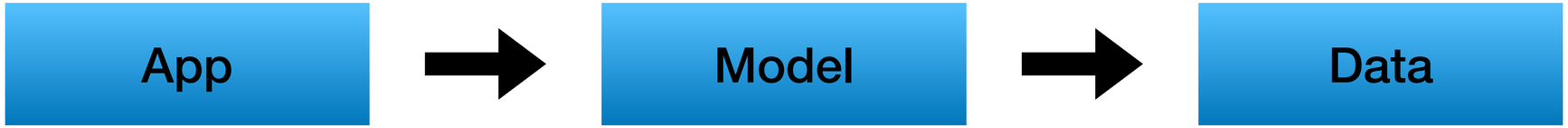
```
SELECT *
FROM users_by_country
WHERE country = ‘Israel’
```

TLDR (2)

- Relational
  focus on entities



- NoSQL
  focus on queries



Modeling is a Science

- Tested methodologies
- Reproducible

Modeling is an Art

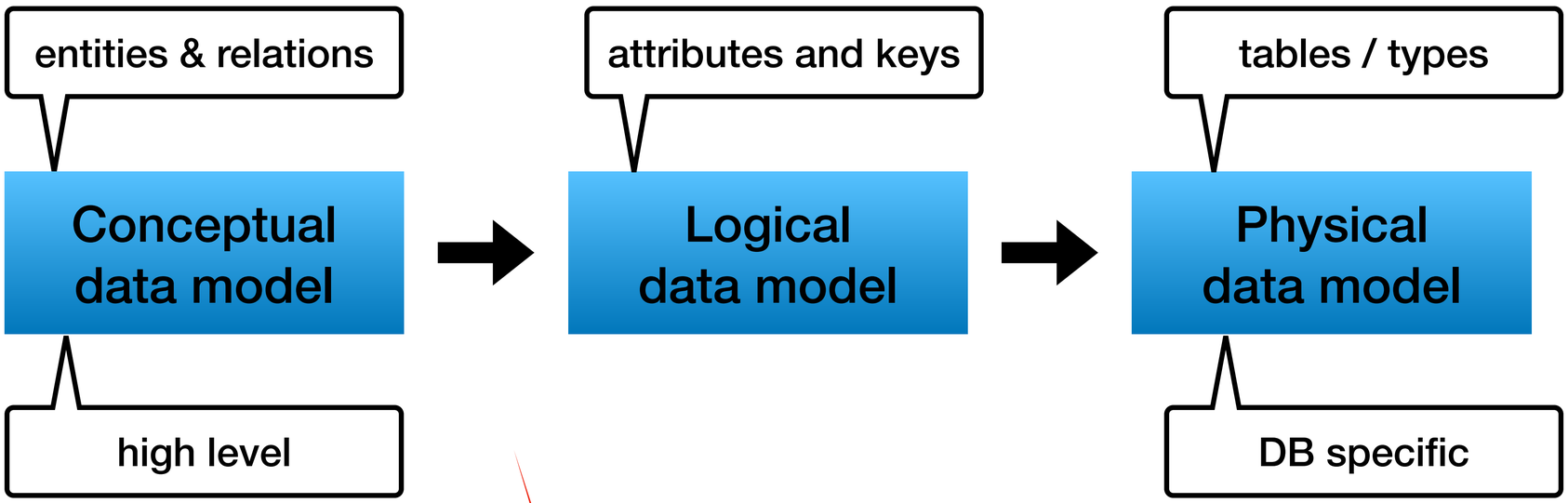
- Multiple ways to solve design problems
- Uncommon use case —> think out of the box

Data modeling process

- Option A: start creating tables
  - run fast and hope for the best

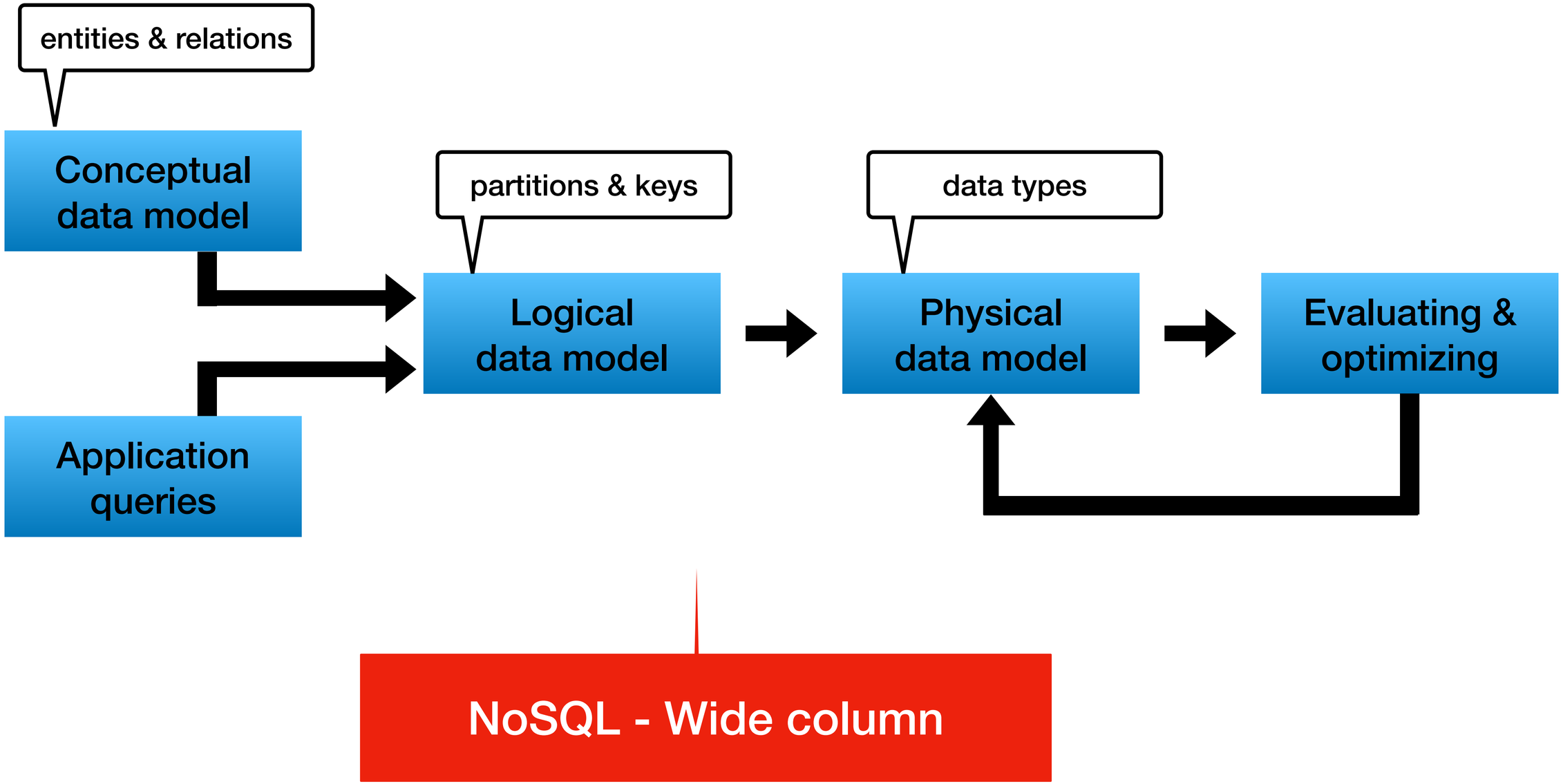
- Option B: follow the modeling process
  - looks time consuming but in practice is faster
  - all team members can help

Data modeling - 10,000 foot view

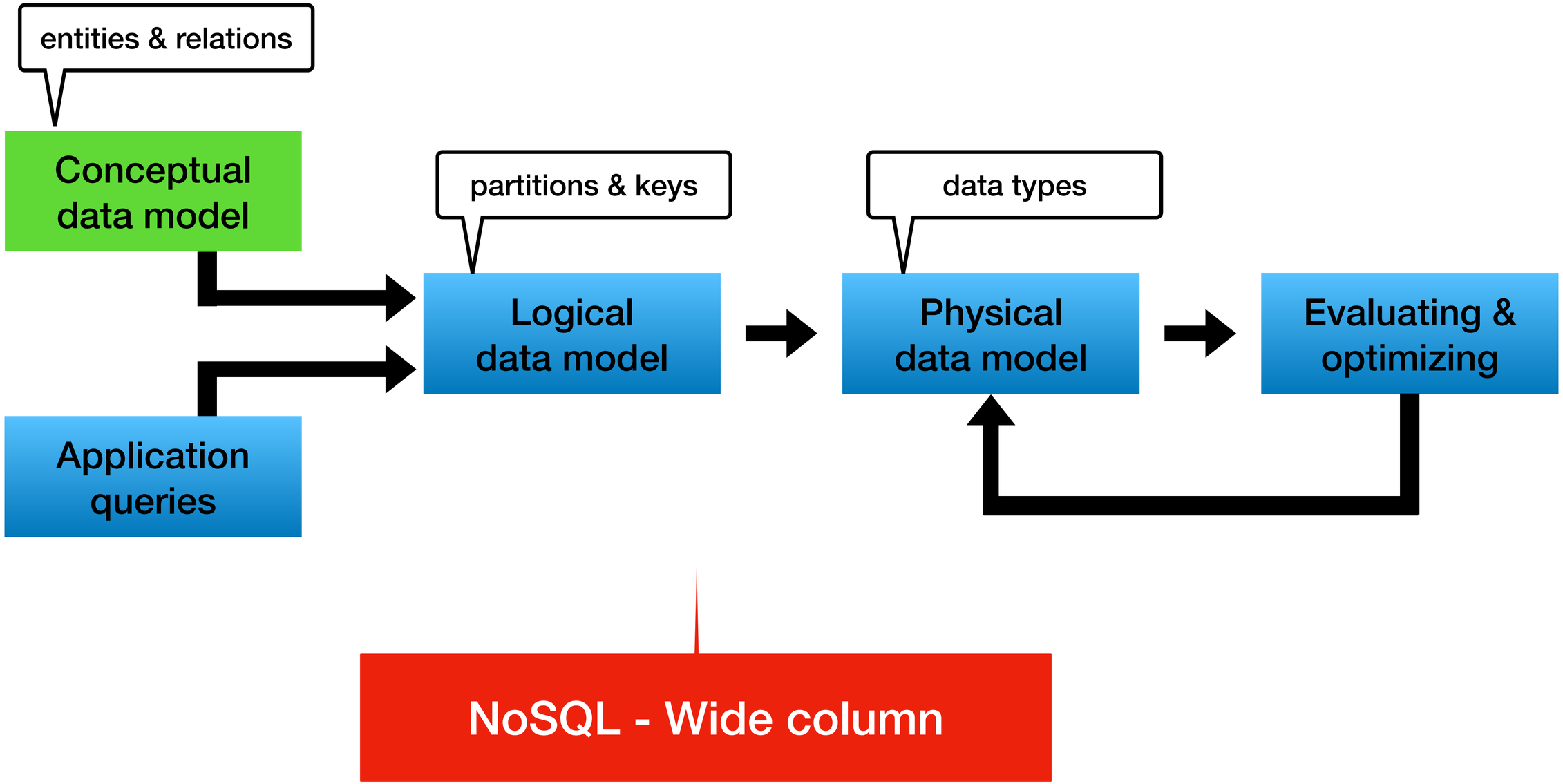


REMINDER - RELATIONAL DATABASE

Data modeling - 10,000 foot view



Data modeling - 10,000 foot view

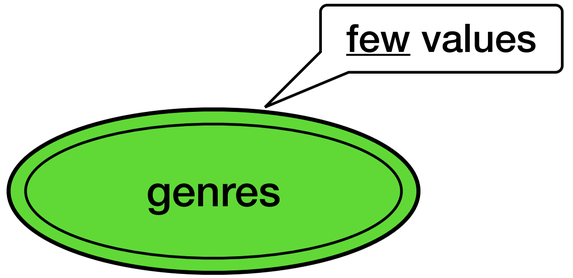


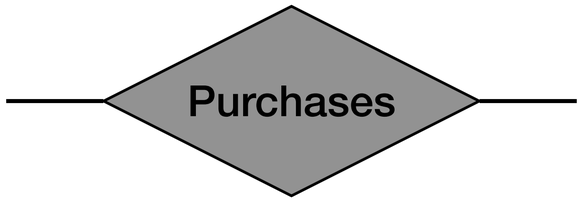
Conceptual data modeling

- Abstract view of entities and relations
- ER Model
  (entity-relation model)
- Same (*) as for relational databases
- “Independent” from specific DB

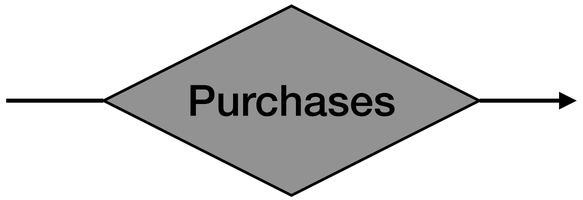
ER Model

- Entities
- Attributes
- Relations
  between entities

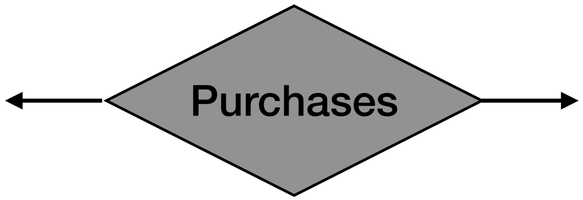




many-many



many-one

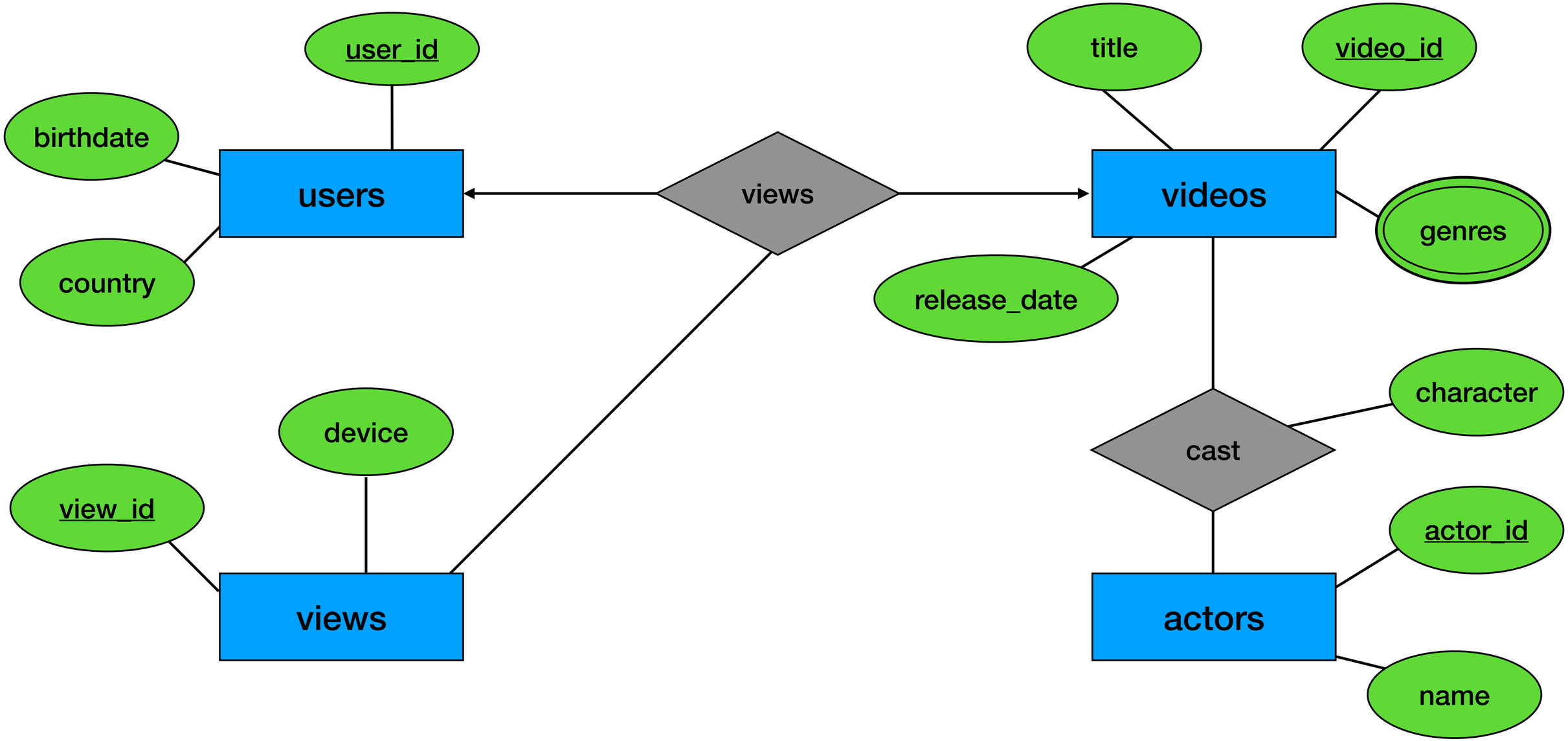


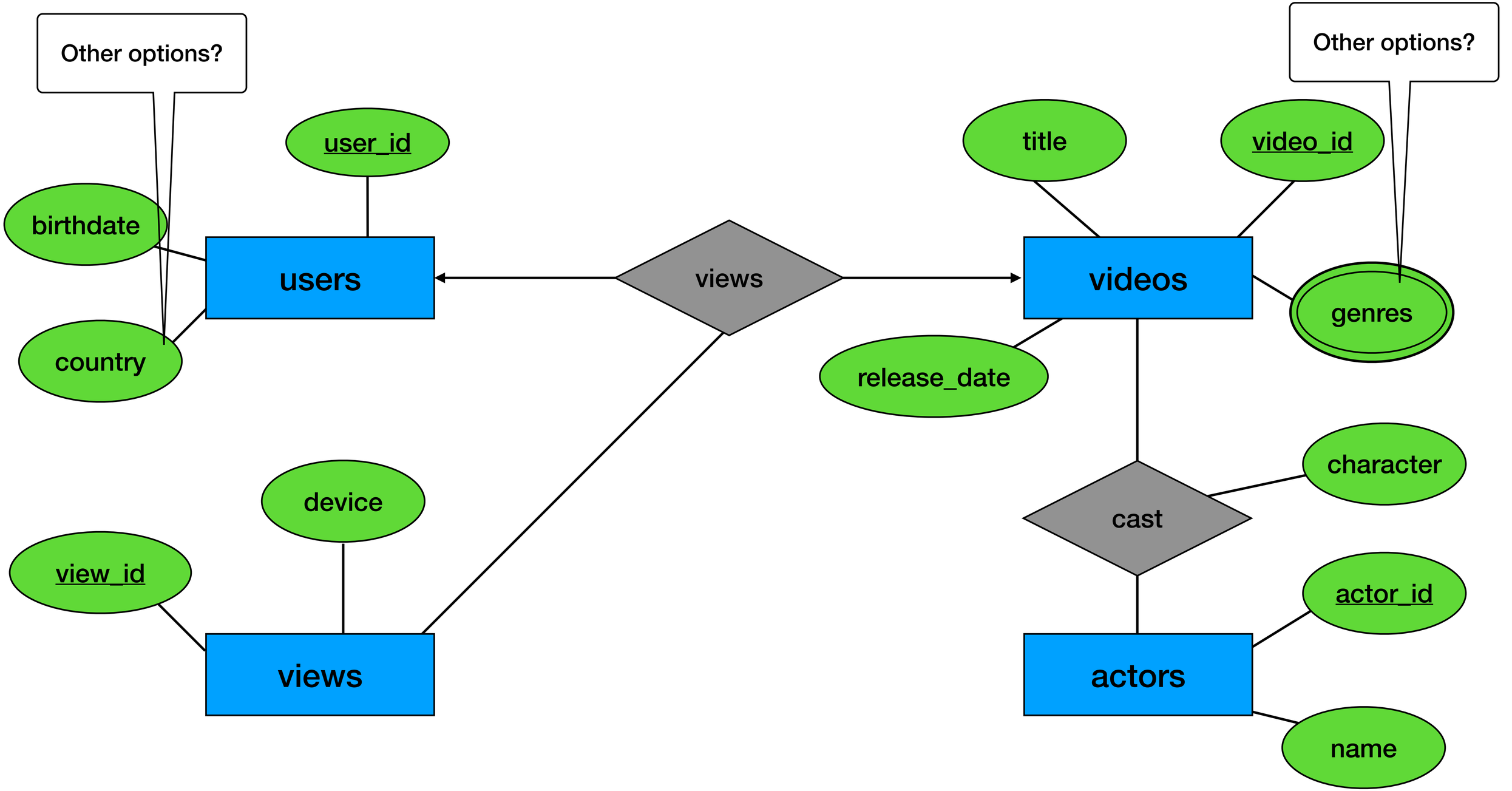
one-one

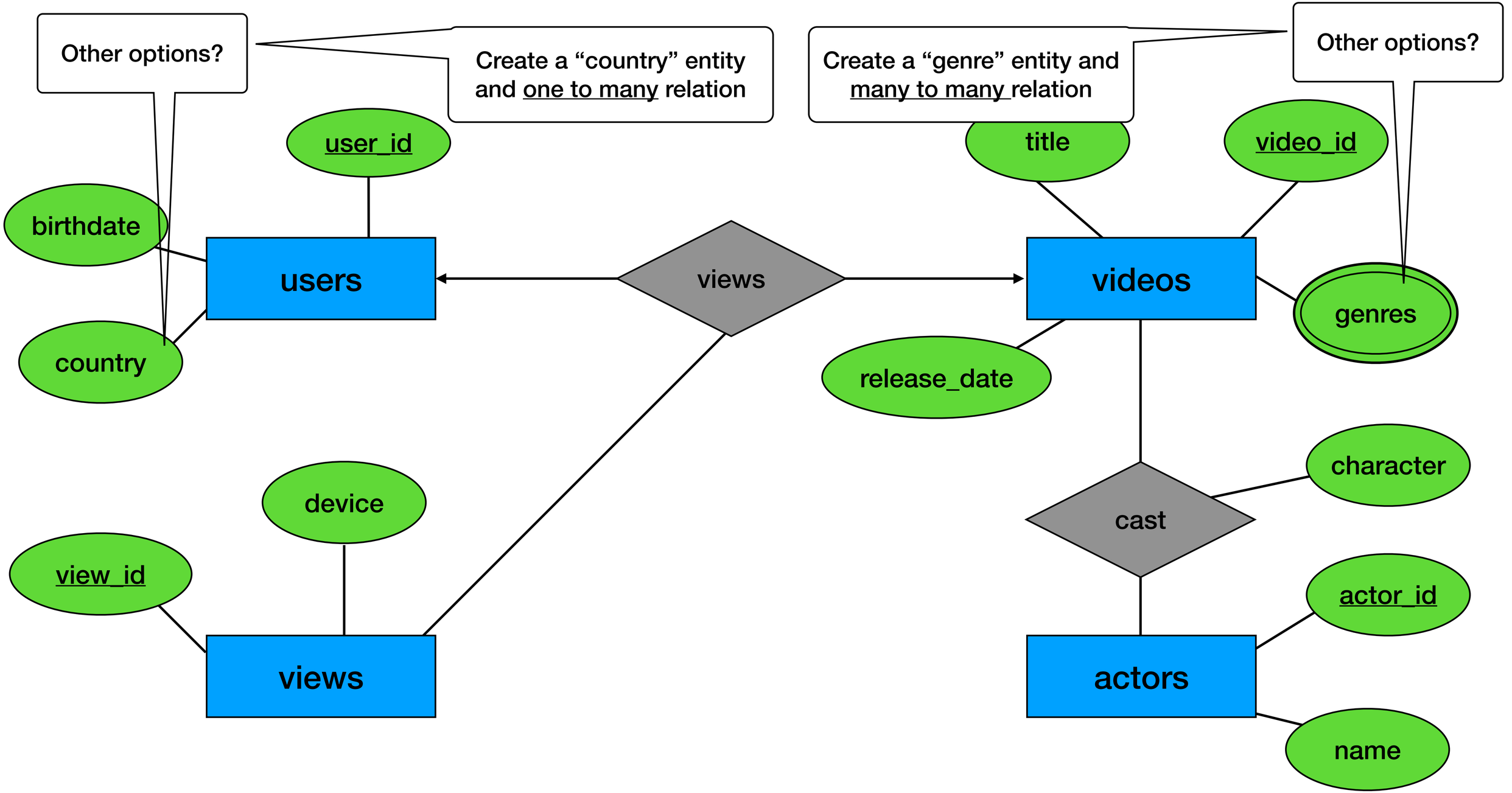
* There are more types like ISA (is a)

Example use case

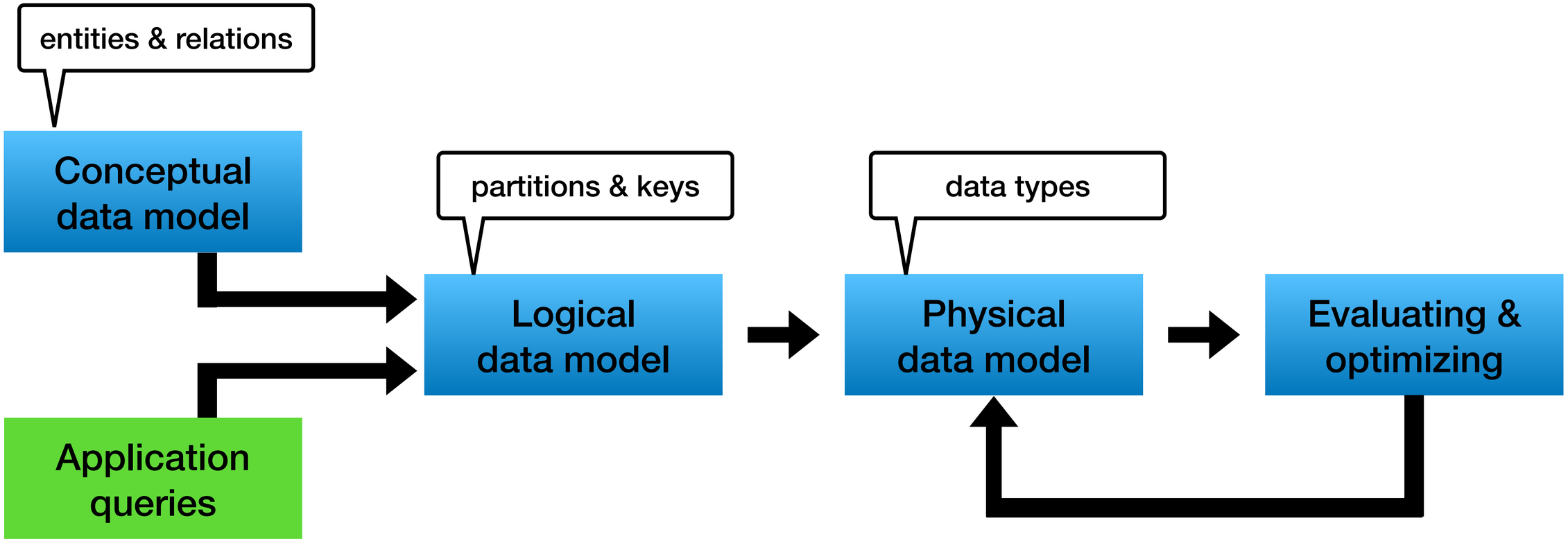
- We are building a simple video streaming service


user_id
birthdate
users
country
views
title
video_id
videos
genres
release_date
device
view_id
views
character
cast
actor_id
actors
name


Other options?
user_id
birthdate
users
country
views
title
video_id
Other options?
videos
genres
release_date
character
device
cast
view_id
actor_id
views
actors
name


Other options?
Create a “country” entity and one to many relation
Create a “genre” entity and many to many relation
Other options?
user_id
title
video_id
birthdate
users
views
videos
genres
country
release_date
device
character
cast
view_id
actor_id
views
actors
name

Data modeling - 10,000 foot view



Application queries

- Goal: model the application workflow
- Not only client workflow
  recommendation engine for example
- Defined by queries

This is given. Usually by both the product and the backend teams

Application queries - example

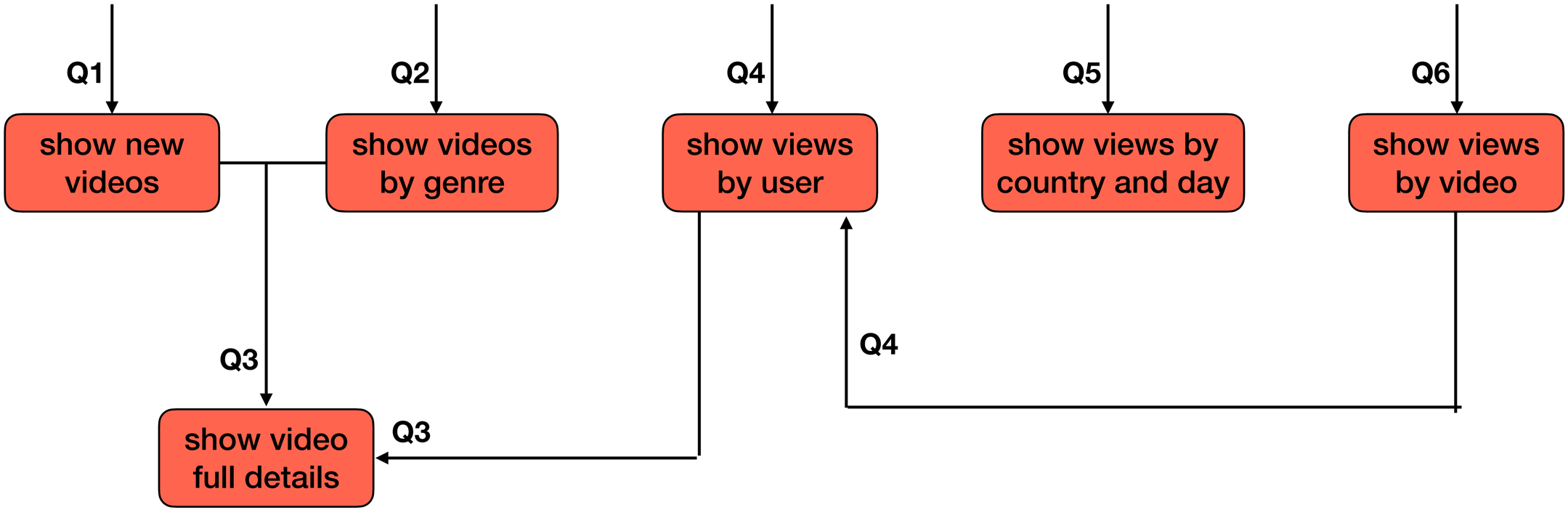
Client workflow

- Q1: Show new videos
- Q2: Show videos by a genre
- Q3: Show video full details

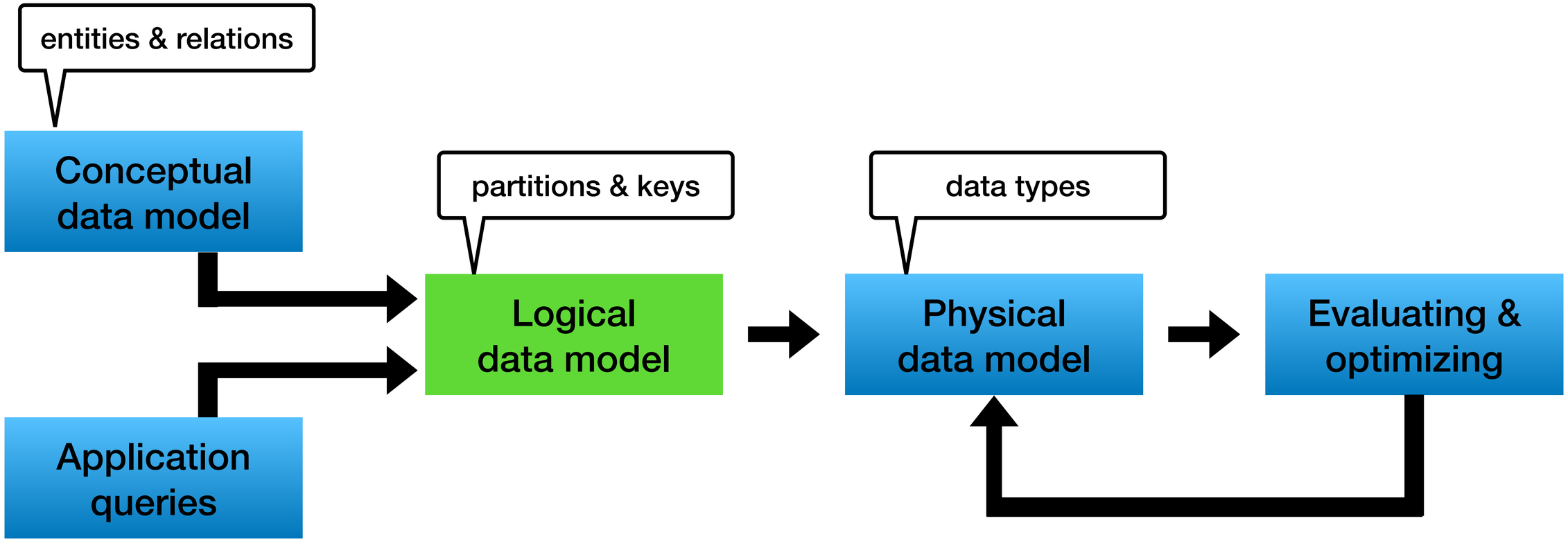
For recommendation engine (online/offline workflow)

- Q4: Show views by user
  watch again / continue watching
- Q5: Show views by country and day
  regional trending
- Q6: Show views by video
  people who watch X also watched

Application workflow - example



Data modeling - 10,000 foot view



Logical data model

Mapping conceptual and queries to tables:

1. queries —> tables
   use “by” convention (for example `users_by_country`)

2. **Identify primary keys**
   partition key columns and clustering columns

3. Add additional attributes

unlike relational DBs,

entities / relations does not convert to tables automatically

* Image from Wikipedia

Chebotko diagrams notation

table_name	
column_1	K
column_2	▼C
column_3	▲C
column_4	S
column_5	++
[column_6]	
{column_7}	
<column_8>	
column_9	
column_10	

Partition key

Clustering key (desc)

Clustering key (asc)

Static column

Counter

List

Set

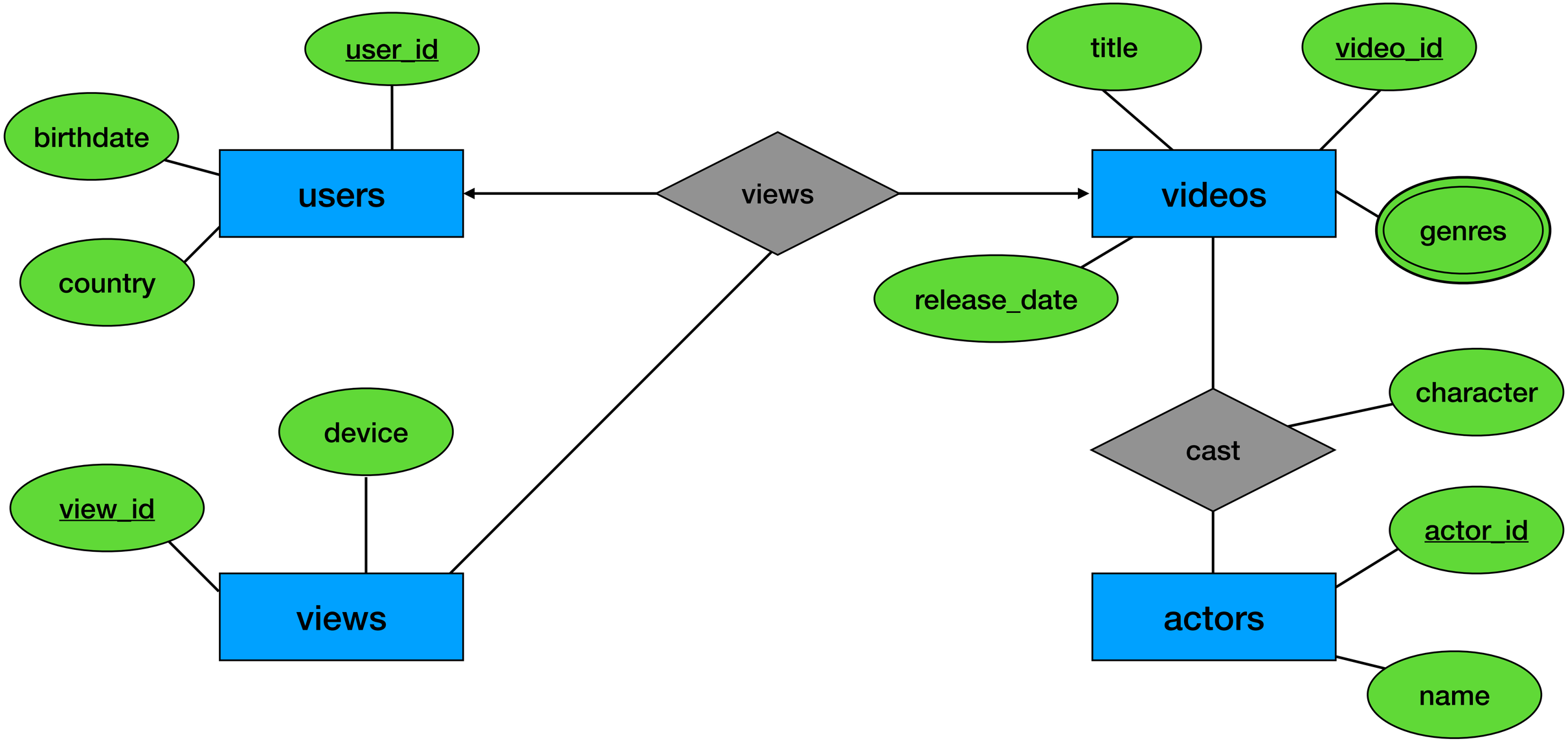
Map

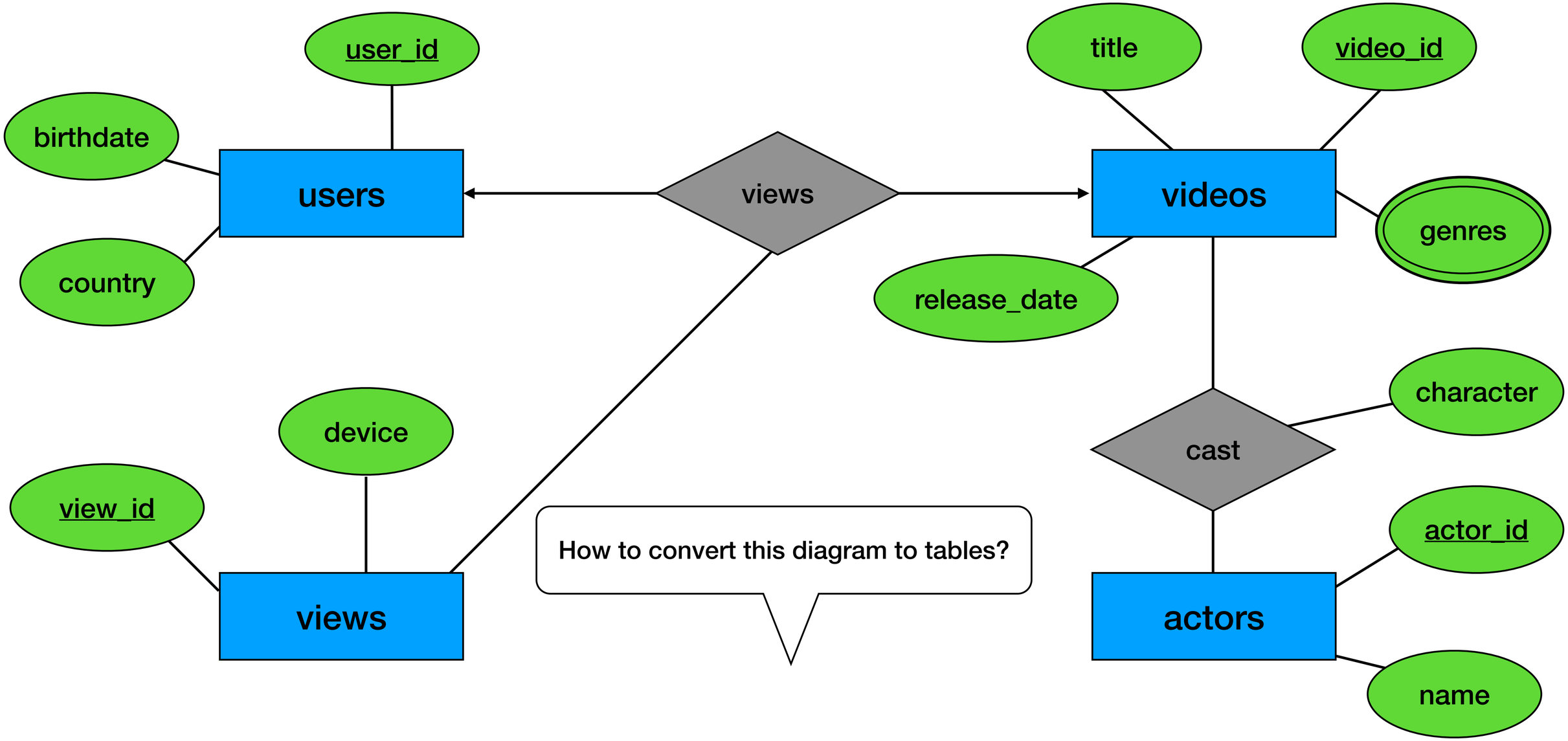
UDT

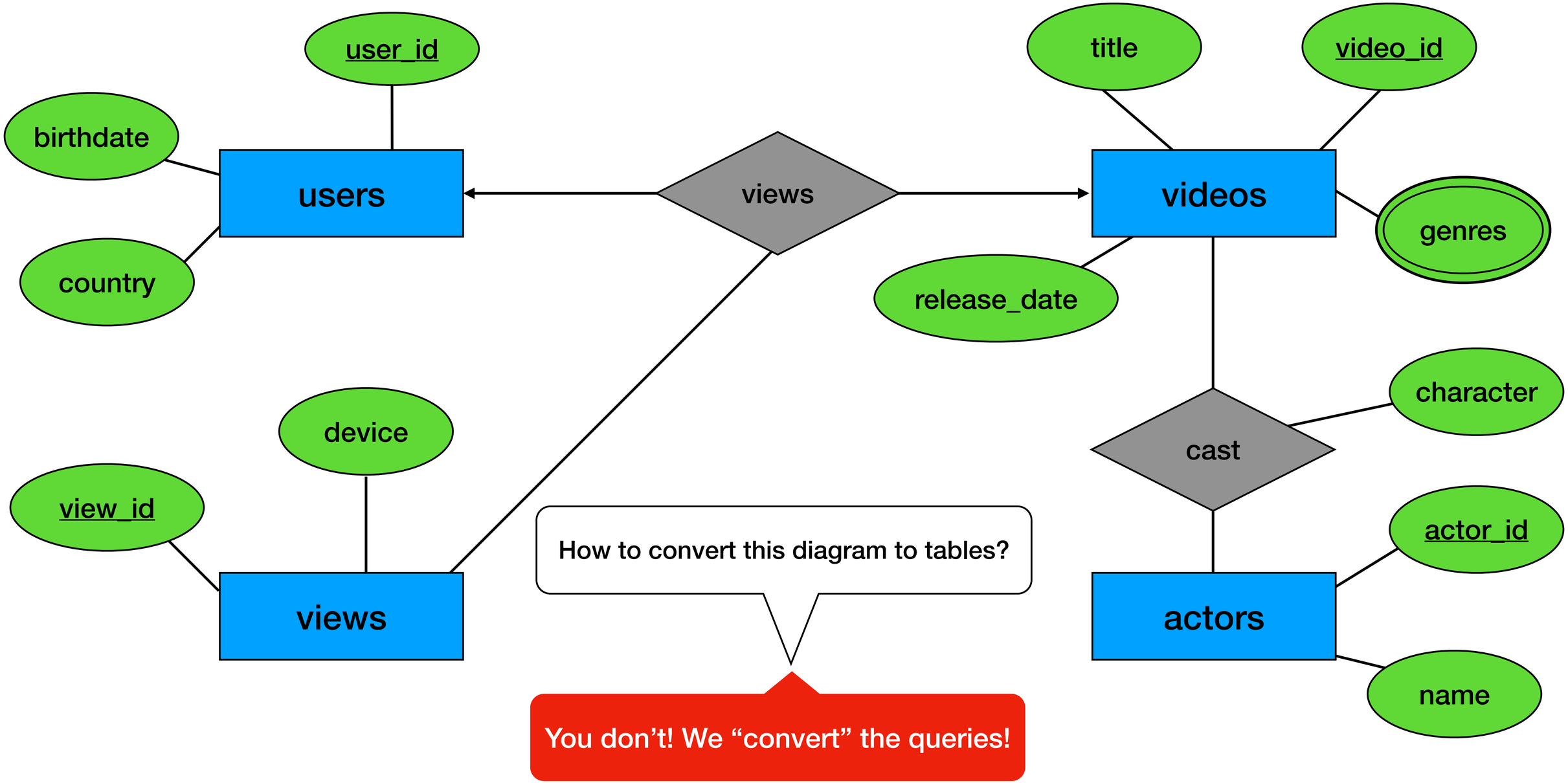
Regular column

* Image from Wikipedia

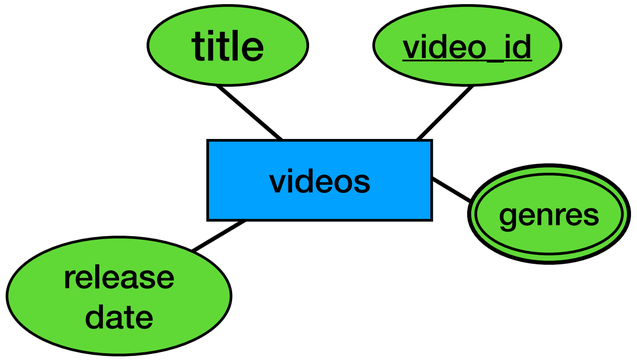
Chewbacca != Chebotko


user_id
birthdate
users
country
views
title
video_id
videos
genres
release_date
device
view_id
views
character
cast
actor_id
actors
name


user_id
birthdate
users
country
views
title
video_id
videos
genres
release_date
device
view_id
views
How to convert this diagram to tables?
character
cast
actor_id
actors
name


user_id
birthdate
users
country
views
title
video_id
videos
genres
release_date
device
view_id
views
How to convert this diagram to tables?
character
cast
actor_id
actors
name
You don’t! We “convert” the queries!

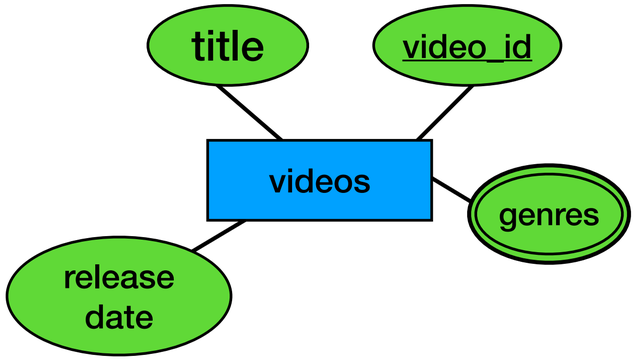
Logical data model - example



Q3

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

Logical data model - example

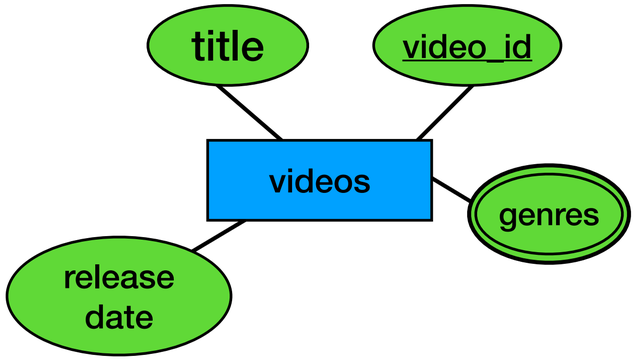


Q3

videos_by_id	
video_id	K
release_date	
title	
{genres}	

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

Logical data model - example



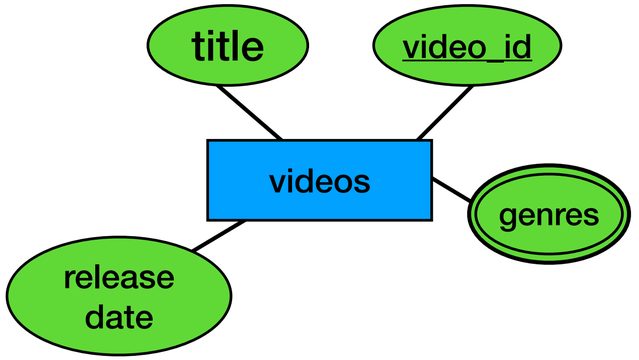
Q3

videos_by_id	
video_id	K
release_date	
title	
{genres}	

Q1

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

Logical data model - example



Q3

videos_by_id	
video_id	K
release_date	
title	
{genres}	

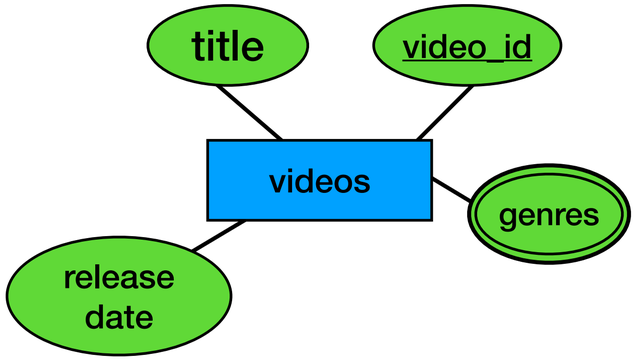
Q1

videos_by_releasedate	
release_date	K
video_id	
title	

Application logic - title is needed but genres are not

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

Logical data model - example



Q3

videos_by_id	
video_id	K
release_date	
title	
{genres}	

Q1

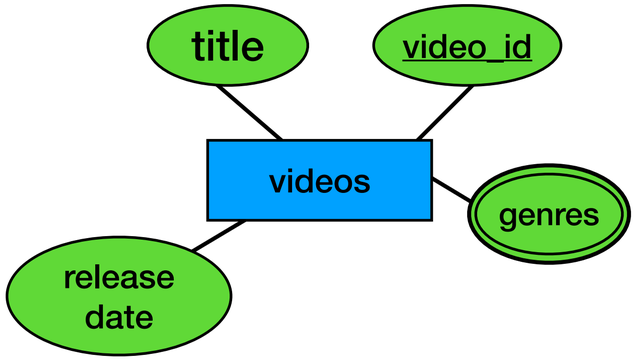
videos_by_releasedate	
release_date	K
video_id	
title	

Application logic - title is needed but genres are not

What do you think?

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

Altering the partition key



Q3

videos_by_id	
video_id	K
release_date	
title	
{genres}	

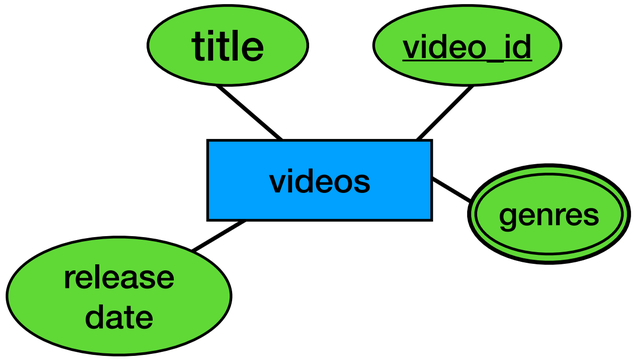
Q1

videos_by_releasedate	
release_year	K
release_date	▼
video_id	
title	

Year / month / anything else
depends on the application logic

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

Altering the partition key



Q3

videos_by_id	
video_id	K
release_date	
title	
{genres}	

Q1

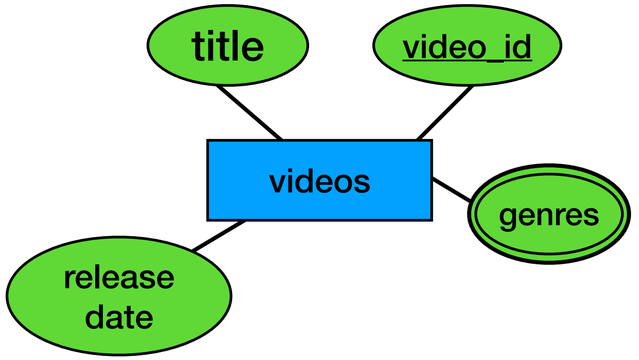
videos_by_releasedate	
release_year	K
release_date	▼
video_id	
title	

Next week we will add to the discuss the size of the partition

Year / month / anything else depends on the application logic

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

Altering the partition key



Q3

videos_by_id	
video_id	K
release_date	
title	
{genres}	

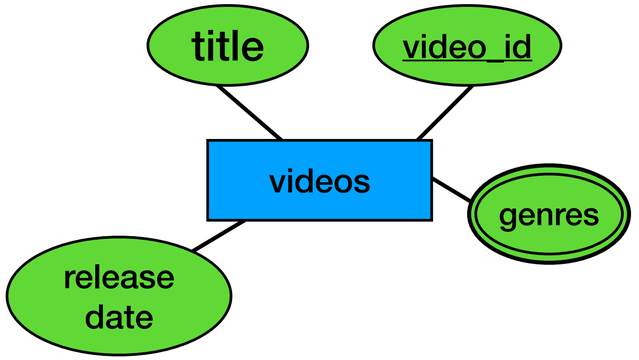
Q1

videos_by_releasedate	
release_year	K
release_date	▼
video_id	
title	

Is this ok?

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

Altering the clustering columns



Q3

videos_by_id	
video_id	K
release_date	
title	
{genres}	

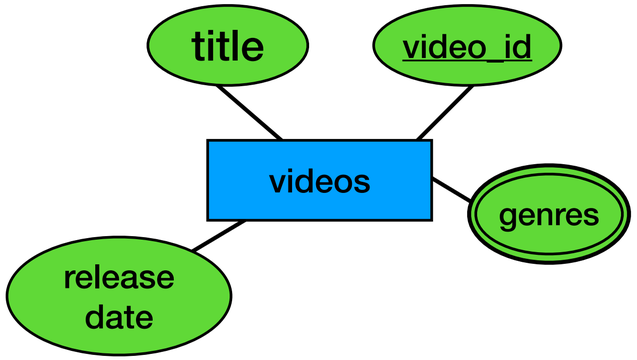
Q1

videos_by_releasedate	
release_year	K
release_date	▼
video_id	▼
title	

Adding uniqueness!

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

More denormalization



Q3

videos_by_id	
video_id	K
release_date	
title	
{genres}	

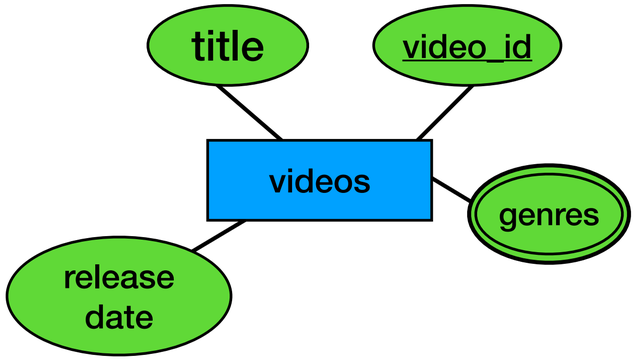
Q1

videos_by_releasedate	
release_year	K
release_date	▼
video_id	▼
title	

Q2

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

More denormalization



Q3

videos_by_id	
video_id	K
release_date	
title	
{genres}	

Q1

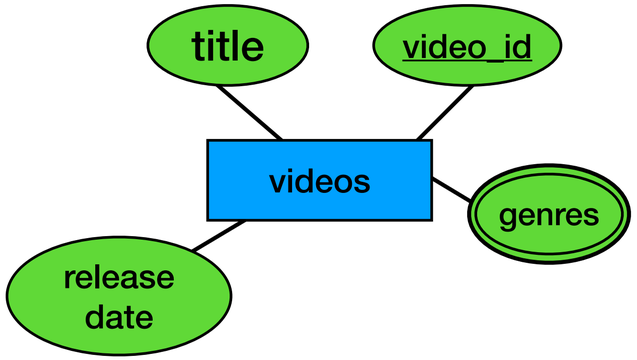
videos_by_releasedate	
release_year	K
release_date	▼
video_id	▼
title	

Q2

videos_by_genre	
genre	K
release_date	▼
video_id	▼
title	

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

More denormalization



This stage is different from “relational modeling” where we had a deterministic algorithm.

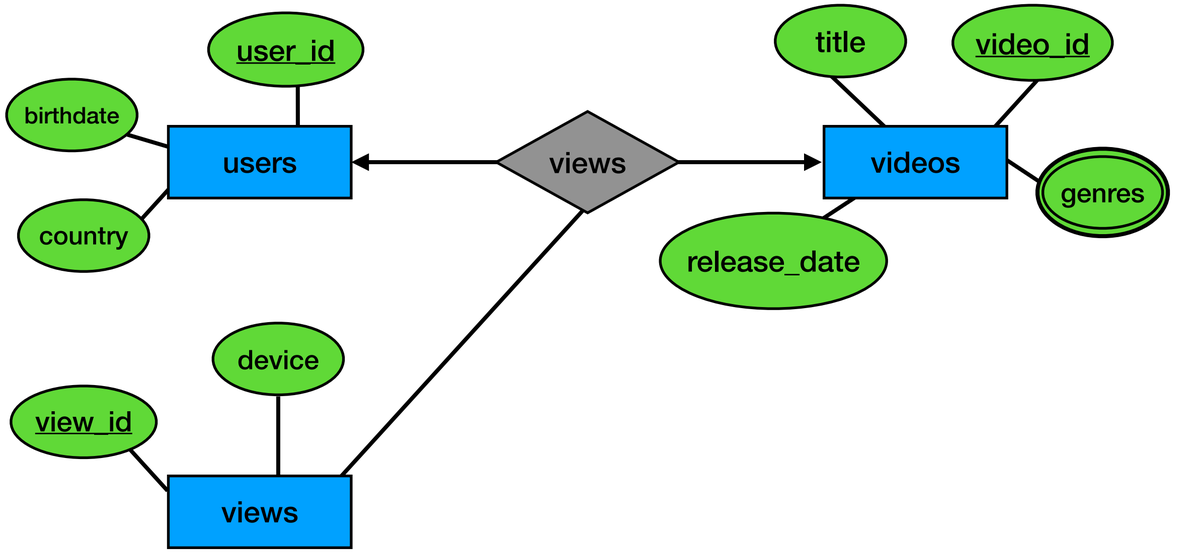
This is why we learned the internals of the system.

Next week we will have more “requirements” to consider (partition size, hot spots…)

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

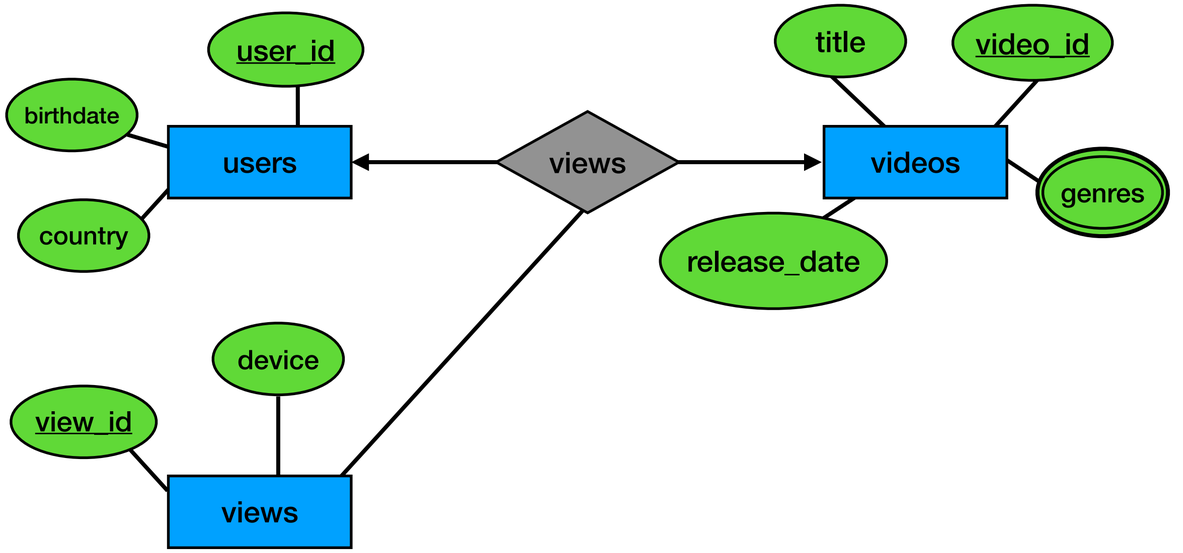
Other queries

Q4



- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

Other queries



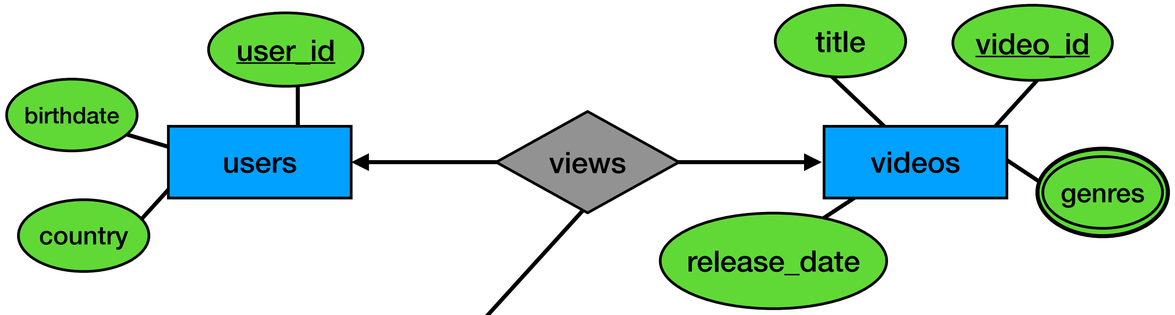
Q4

views_by_user	
user_id	K
view_id	▼
device	
video_id	
title	

We need also the title, not just the video_id (denormalization)

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

Other queries



Q4

views_by_user	
user_id	K
view_id	▼
device	
video_id	
title	

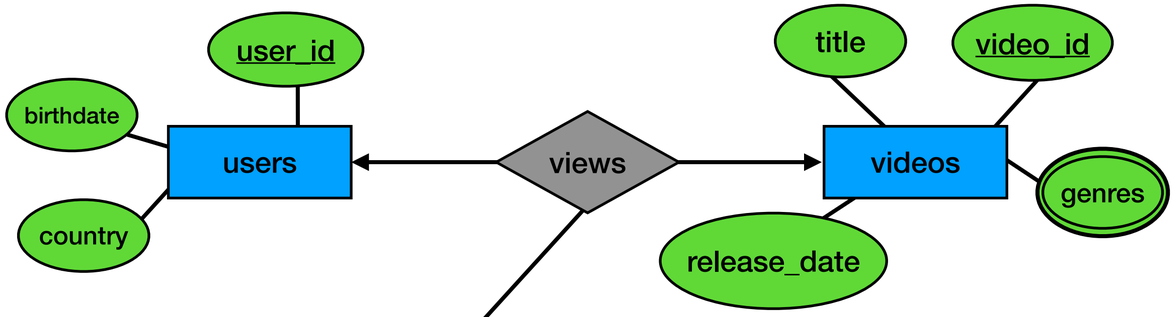
We need also the title, not just the video_id (denormalization)

Side note

In real life - do we store the title?

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

Other queries



Q4

views_by_user	
user_id	K
view_id	▼
device	
video_id	
title	

We need also the title, not just the video_id (denormalization)

Side note

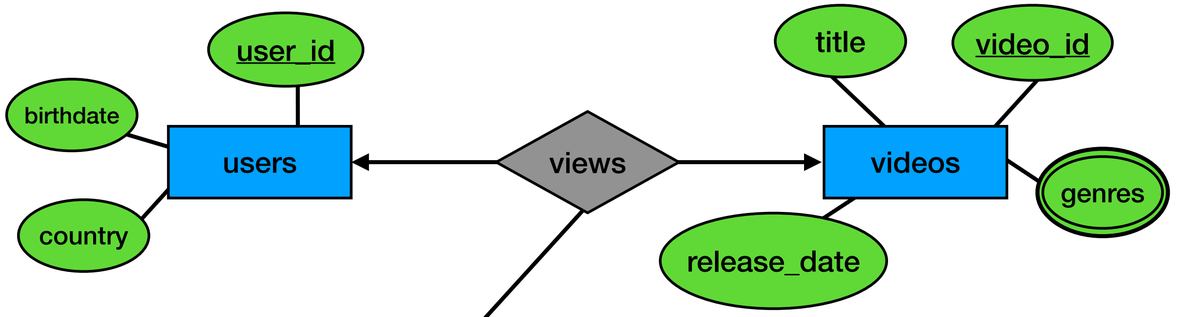
In real life - do we store the title?

Probably not.
(We would not be able to “change” millions of rows of the title changes)

So how would we display the title (or image / description) for a user to view her history?

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

Other queries



Q4

views_by_user	
user_id	K
view_id	▼
device	
video_id	
title	

We need also the title, not just the video_id (denormalization)

Side note

In real life - do we store the title?

Probably not.
(We would not be able to “change” millions of rows of the title changes)

So how would we display the title (or image / description) for a user to view her history?

We would need a caching layer with all the video details
(This is basically a really fast join)

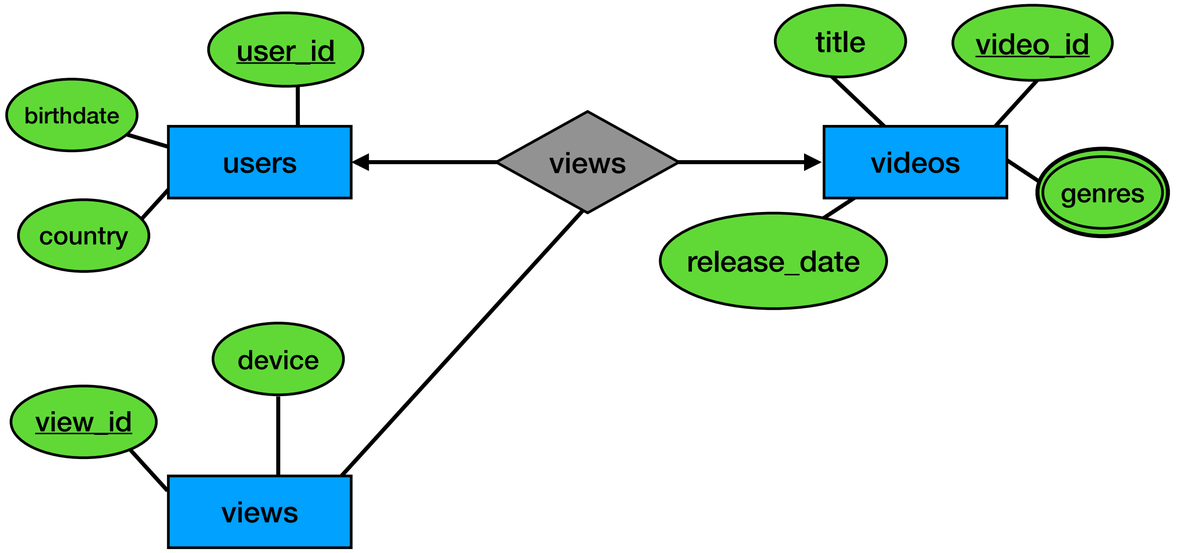
- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: S
- Q5: S
- Q6: S

Other queries

Q4

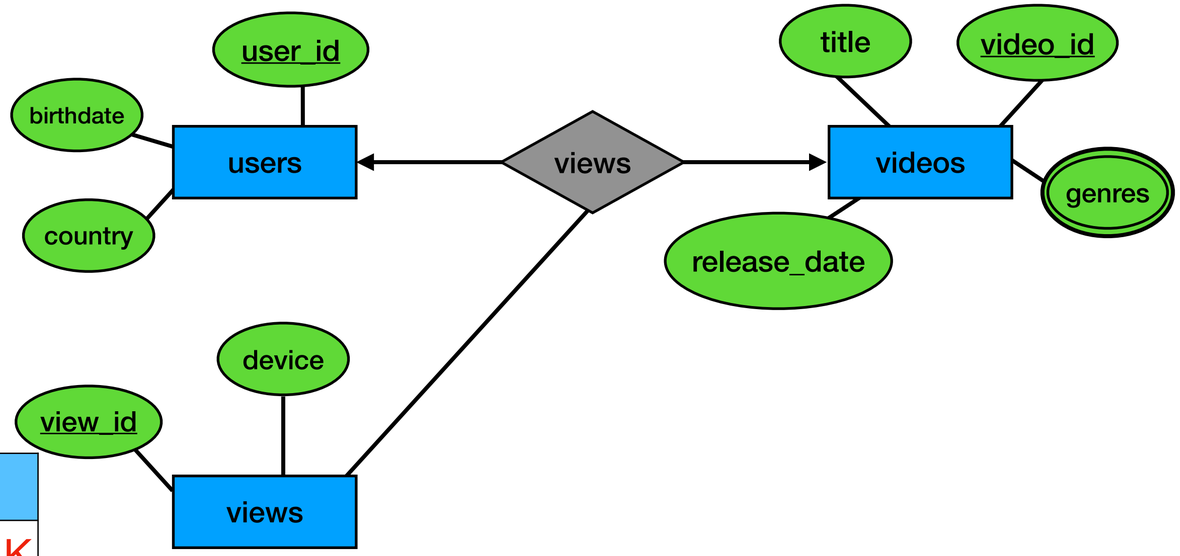
views_by_user	
user_id	K
view_id	▼
device	
video_id	
title	

Q6



- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

Other queries



Q4

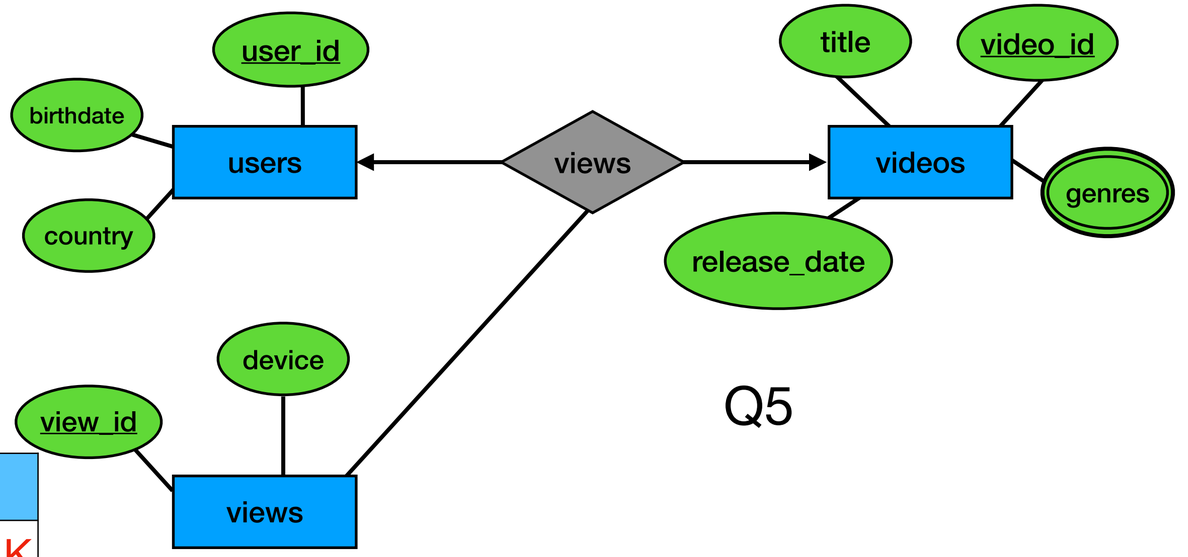
views_by_user	
user_id	K
view_id	▼
device	
video_id	
title	

Q6

views_by_video	
video_id	K
view_id	▼
device	
user_id	

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

Other queries



Q4

views_by_user	
user_id	K
view_id	▼
device	
video_id	
title	

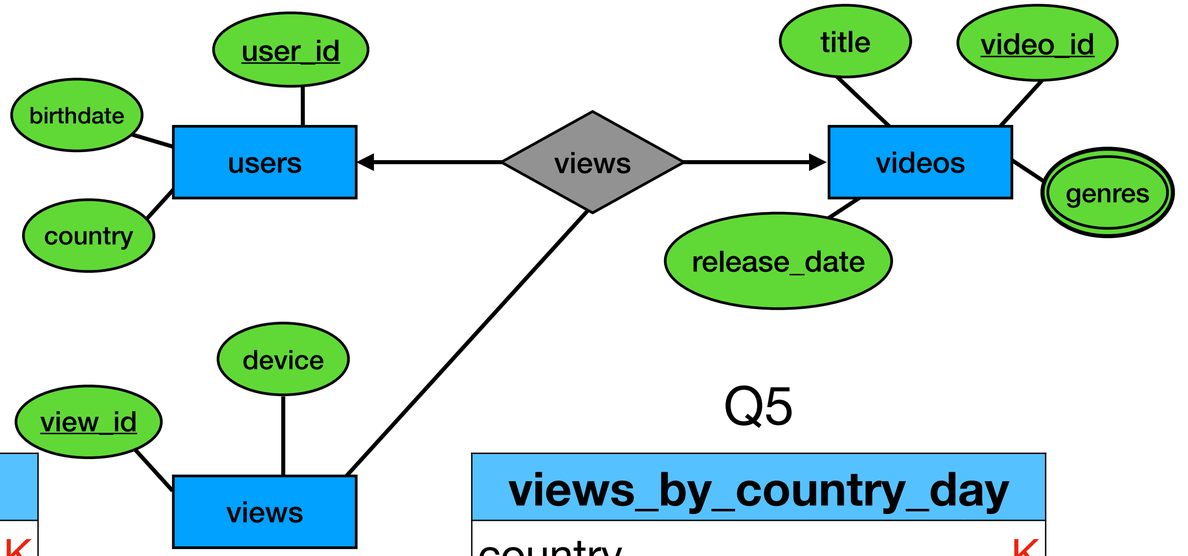
Q6

views_by_video	
video_id	K
view_id	▼
device	
user_id	

Q5

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

Other queries



Q4

views_by_user	
user_id	K
view_id	▼
device	
video_id	
title	

Q6

views_by_video	
video_id	K
view_id	▼
device	
user_id	

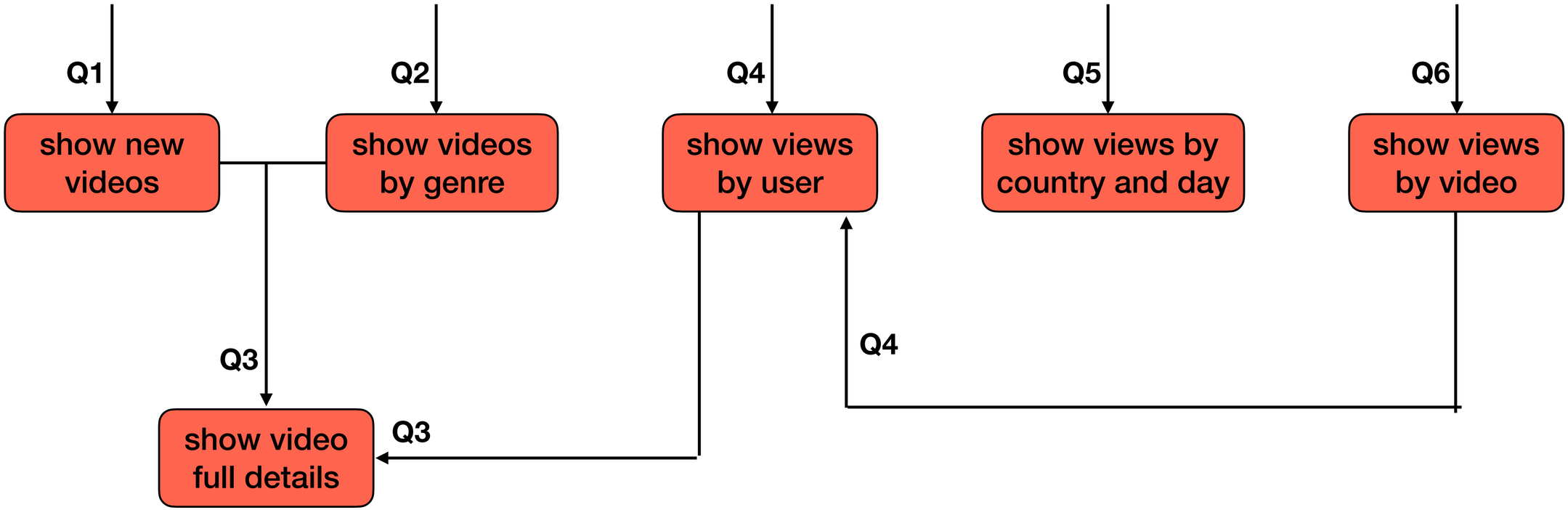
Q5

views_by_country_day	
country	K
day	K
view_id	▼
device	
video_id	
user_id	

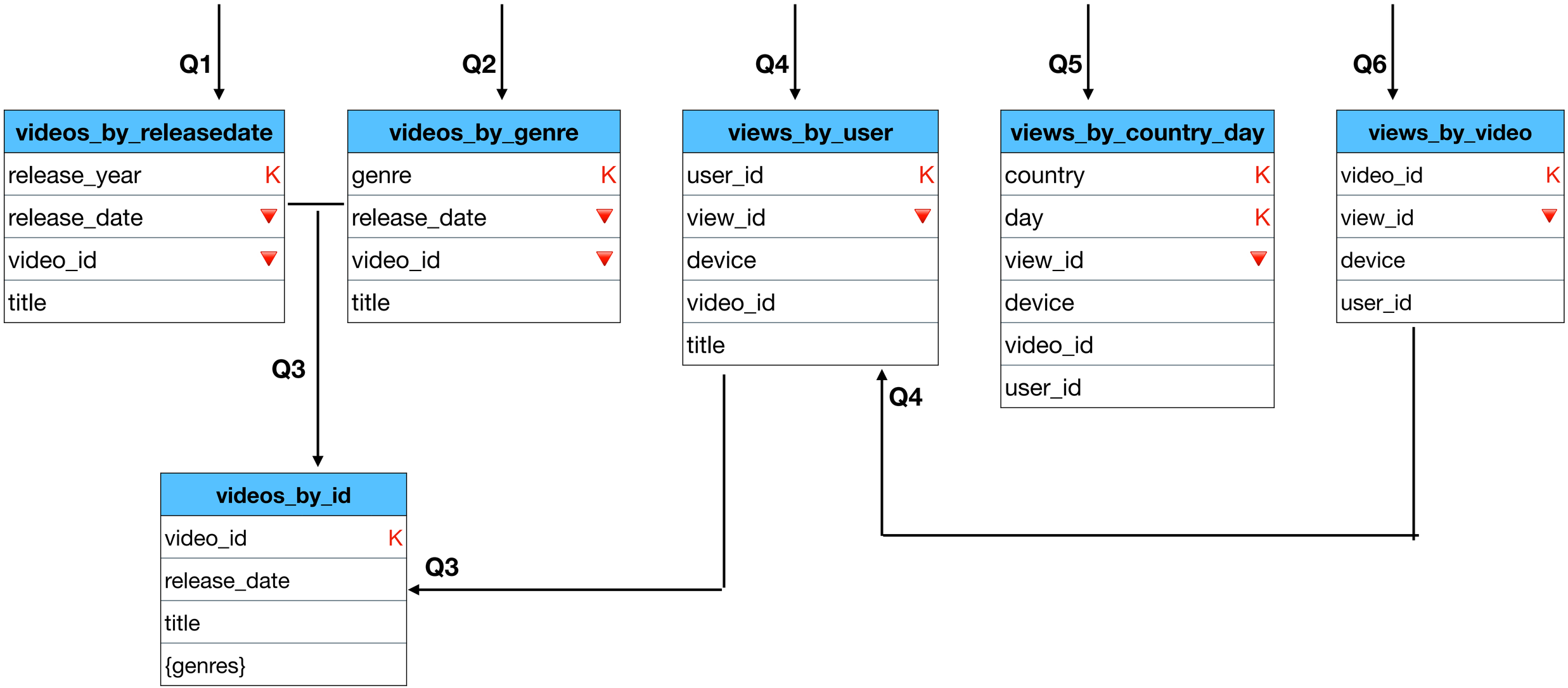
What is the difference between K and ▼ for “day”

- Q1: Show new videos
- Q2: Show videos by genre
- Q3: Show video full details
- Q4: Show views by user
- Q5: Show views by country and day
- Q6: Show views by video

All together



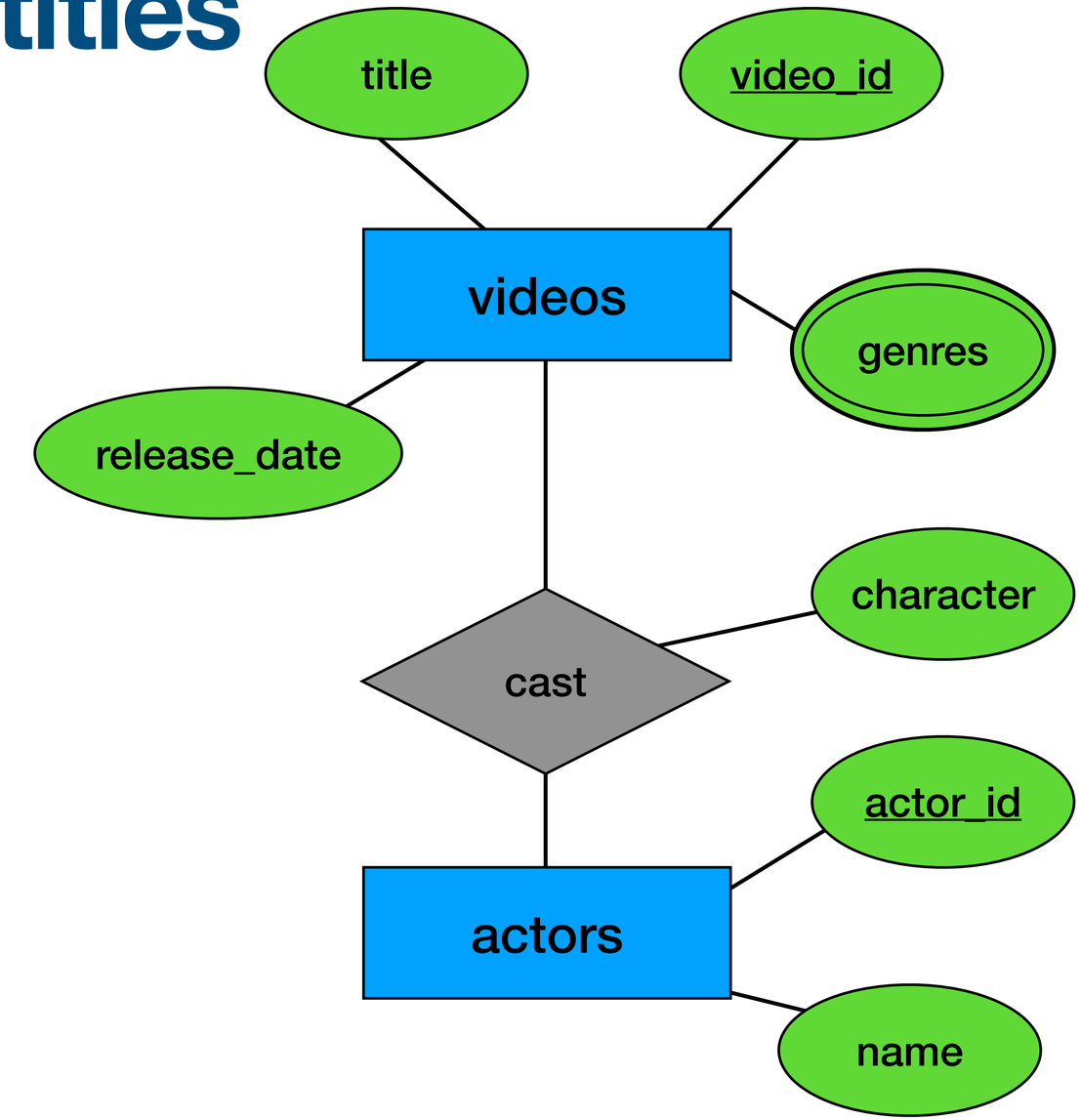
All together



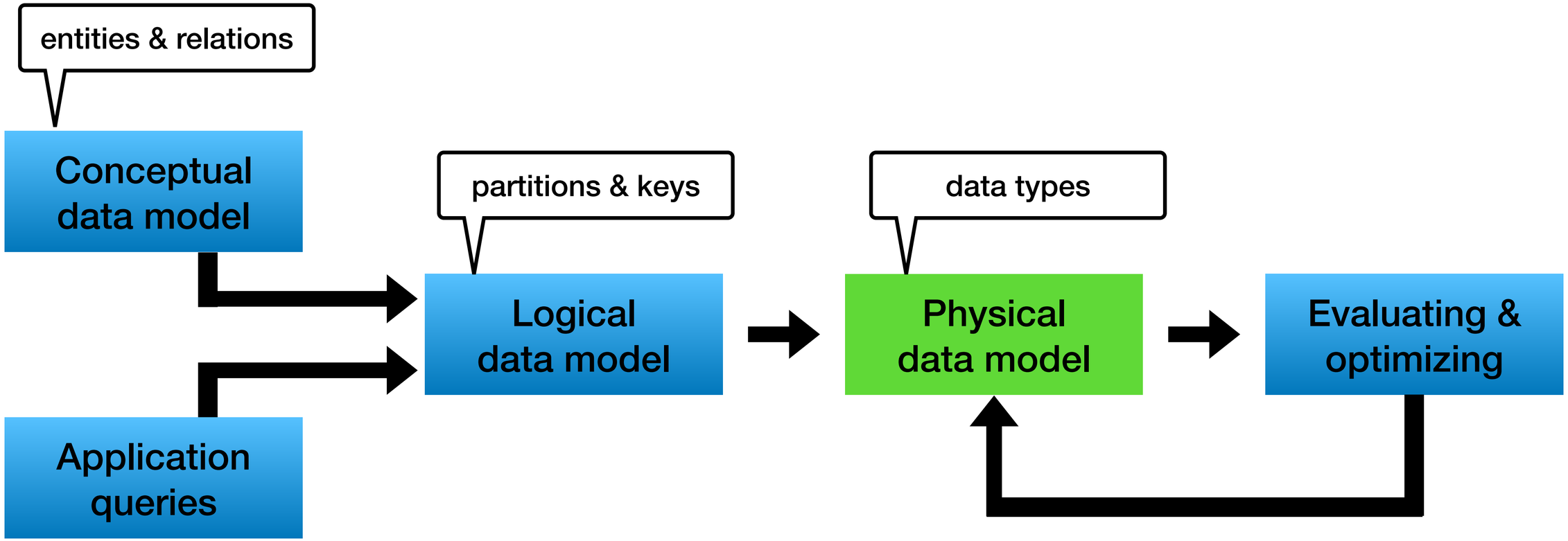
Note about “ghost” entities

- We did not create any table for “actors”
  why?

- Homework: add missing elements so we would create some actor table



Data modeling - 10,000 foot view



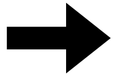
Physical data model

All we have to do:

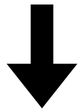
- Add CQL data types
- Add create table statement

Physical data model - example

videos_by_releasedate	
release_year	K
release_date	🔻
video_id	🔻
title	

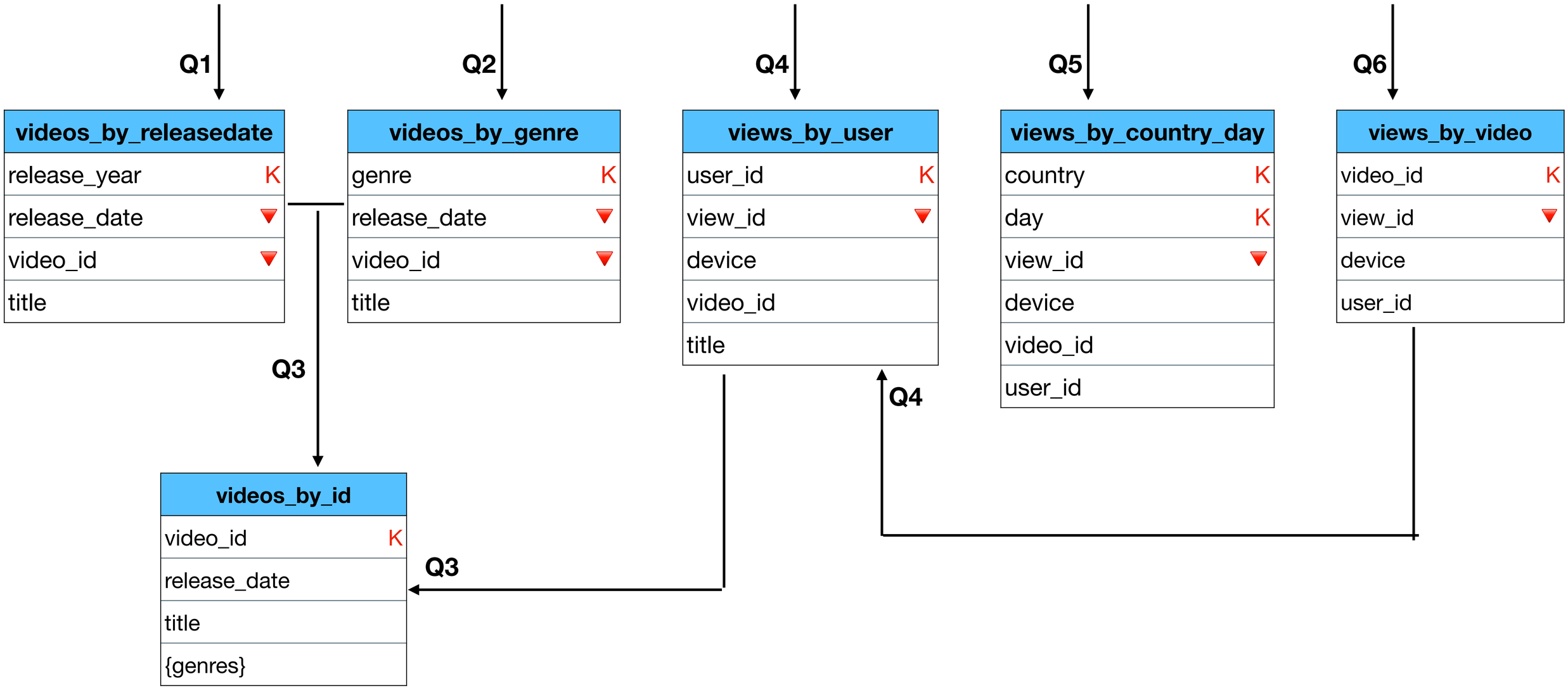


videos_by_releasedate		
release_year	INT	K
release_date	TIMESTAMP	🔻
video_id	BIGINT	🔻
title	TEXT	

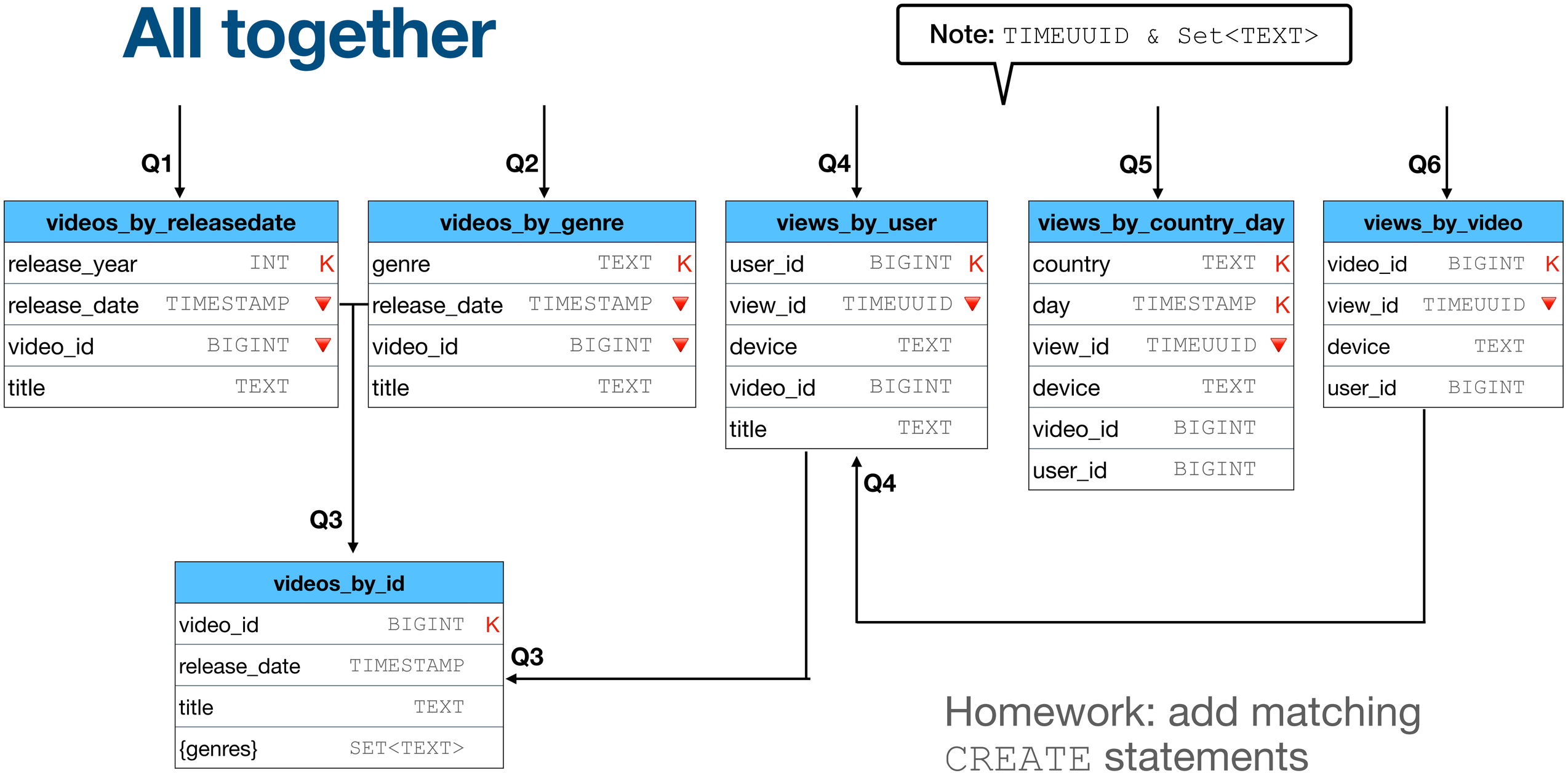


```
CREATE TABLE videos_by_releasedate (
    release_year INT,
    release_date TIMESTAMP,
    video_id     BIGINT,
    title        TEXT,
    PRIMARY KEY ((release_year), release_date, video_id)
) WITH CLUSTERING ORDER BY (release_date DESC, video_id DESC);
```

All together

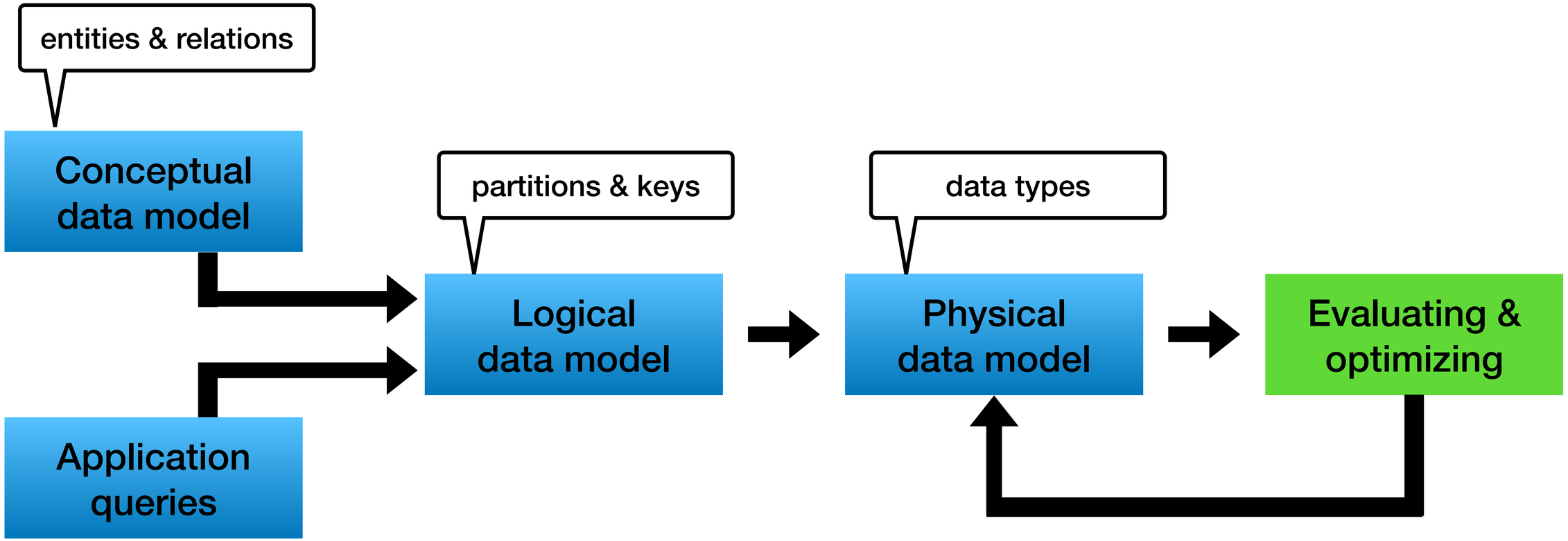


All together



Homework: add matching `CREATE` statements

Data modeling - 10,000 foot view



Evaluating and optimizing

An ongoing process

Usually as you scale

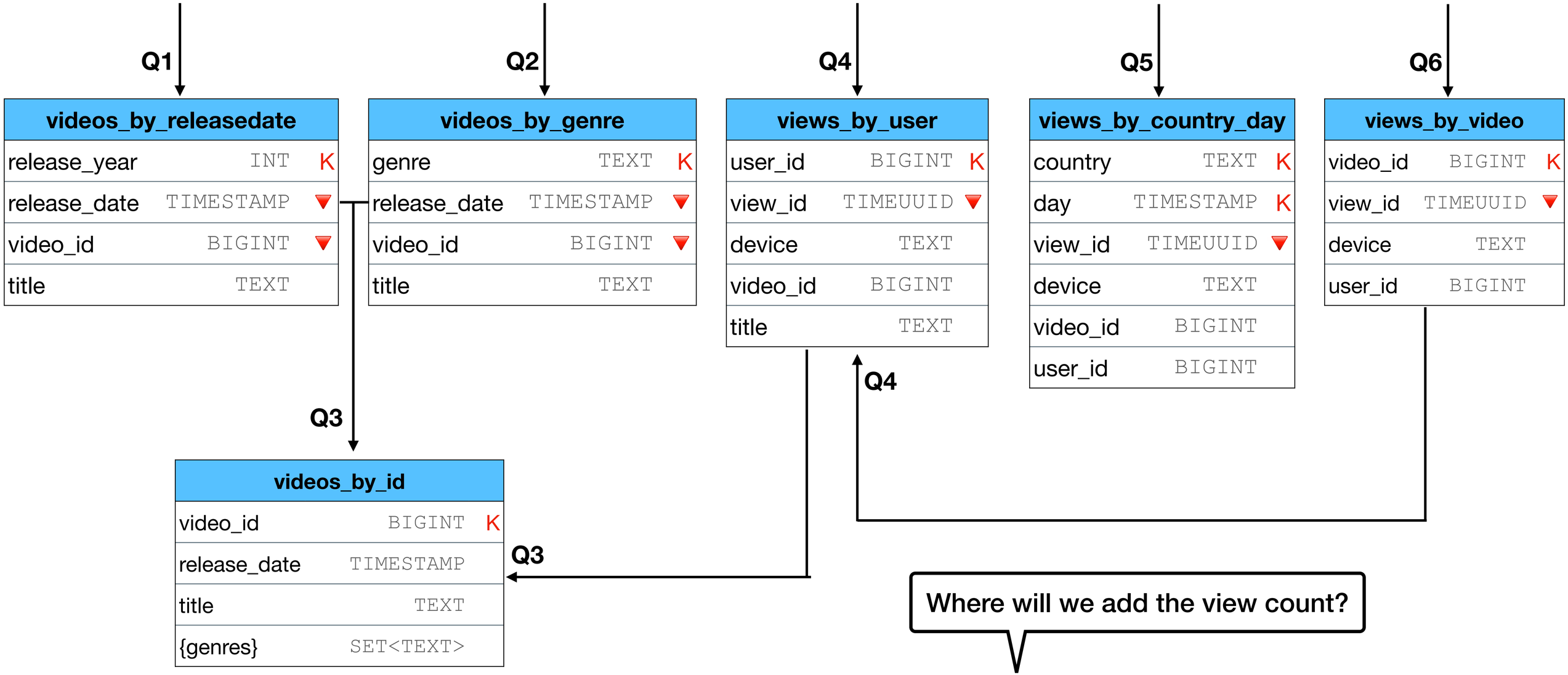
- there are new product requirements
- you find new problems

Evaluating and optimizing - example (1)

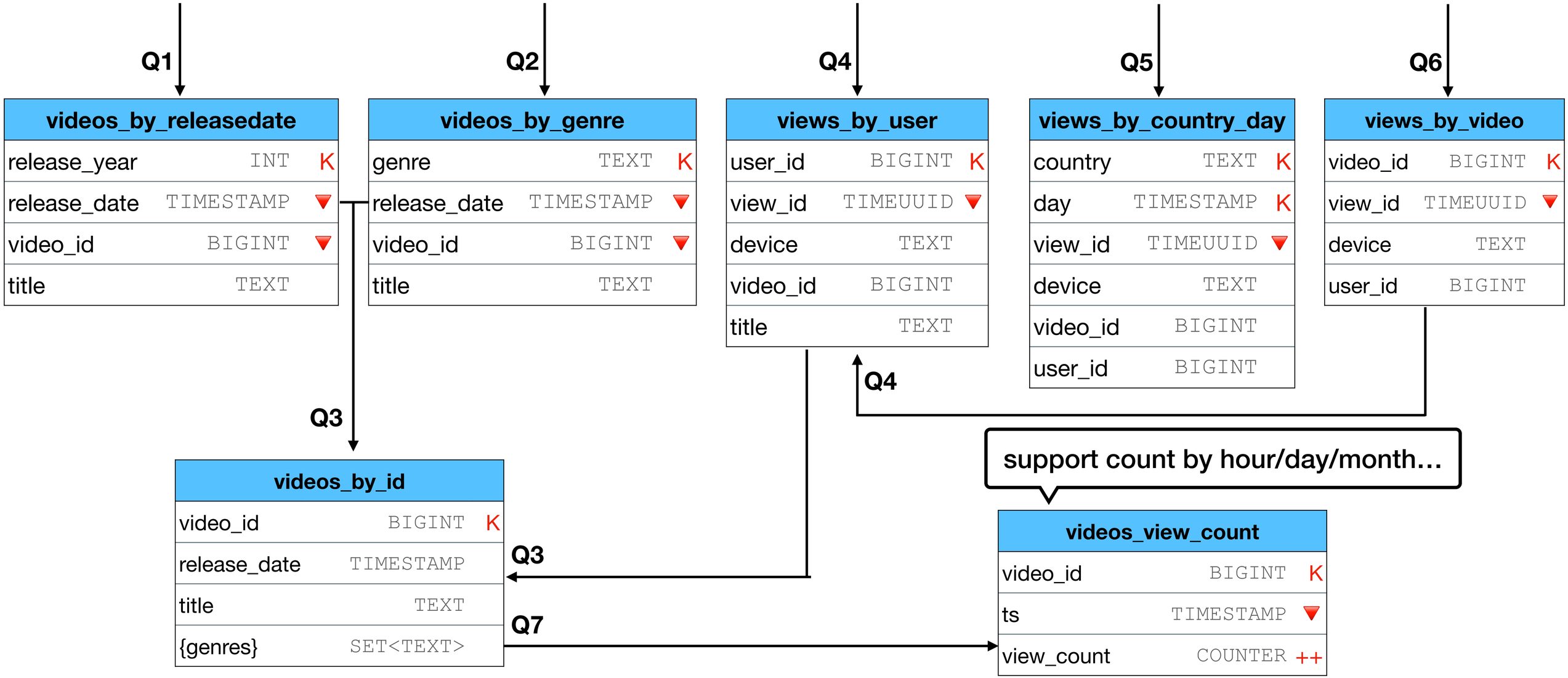
- The streaming service is a big hit
  - More users
  - More usage

The product team requires to add the view count next to each video

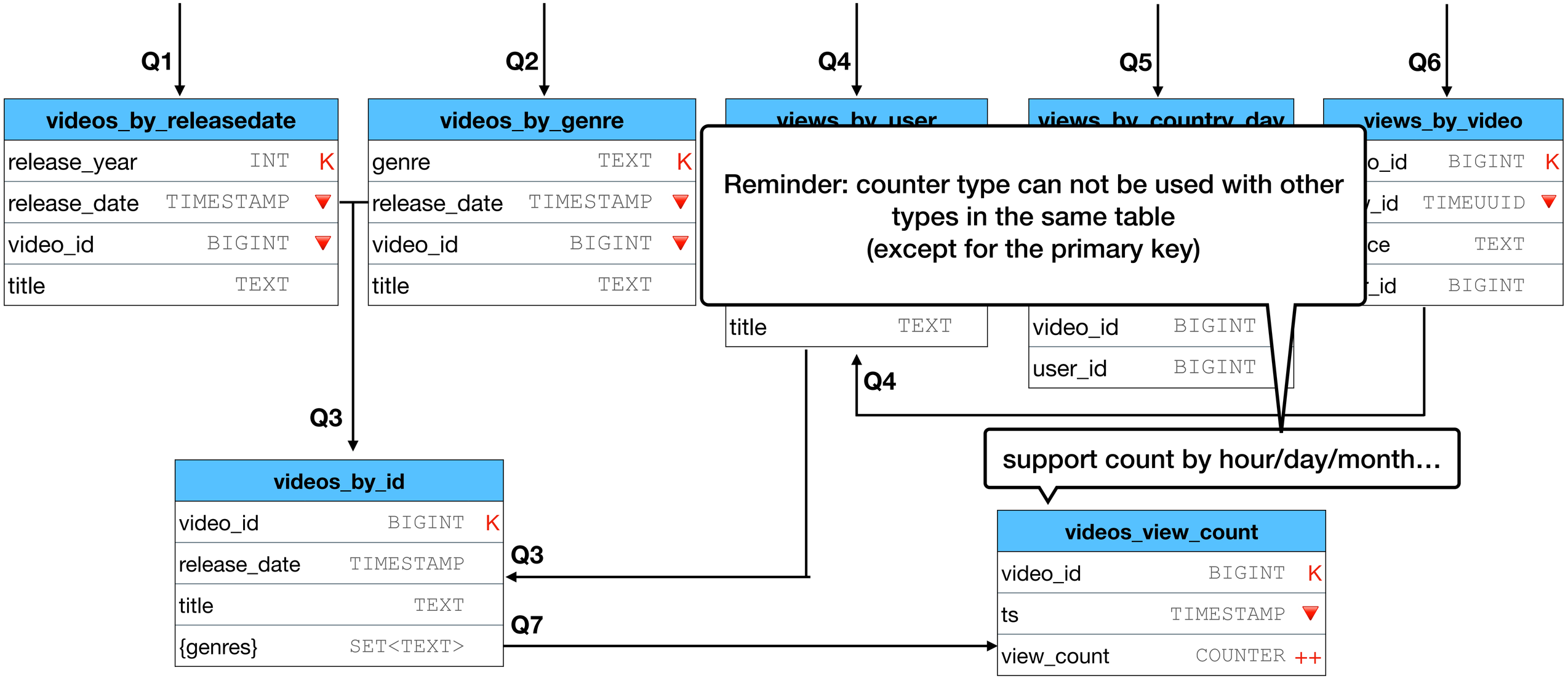
Refining


Q1
Q2
Q4
Q5
Q6
videos_by_releasedate
release_year INT K
release_date TIMESTAMP
video_id BIGINT
title TEXT
videos_by_genre
genre TEXT K
release_date TIMESTAMP
video_id BIGINT
title TEXT
views_by_user
user_id BIGINT K
view_id TIMEUUID
device TEXT
video_id BIGINT
title TEXT
views_by_country_day
country TEXT K
day TIMESTAMP K
view_id TIMEUUID
device TEXT
video_id BIGINT
user_id BIGINT
views_by_video
video_id BIGINT K
view_id TIMEUUID
device TEXT
user_id BIGINT
Q3
videos_by_id
video_id BIGINT K
release_date TIMESTAMP
title TEXT
{genres} SET<TEXT>
Q3
Q4
Where will we add the view count?

Refining


Q1
Q2
Q4
Q5
Q6
videos_by_releasedate
release_year INT K
release_date TIMESTAMP
video_id BIGINT
title TEXT
videos_by_genre
genre TEXT K
release_date TIMESTAMP
video_id BIGINT
title TEXT
views_by_user
user_id BIGINT K
view_id TIMEUUID
device TEXT
video_id BIGINT
title TEXT
views_by_country_day
country TEXT K
day TIMESTAMP K
view_id TIMEUUID
device TEXT
video_id BIGINT
user_id BIGINT
views_by_video
video_id BIGINT K
view_id TIMEUUID
device TEXT
user_id BIGINT
Q3
Q4
videos_by_id
video_id BIGINT K
release_date TIMESTAMP
title TEXT
{genres} SET<TEXT>
Q3
Q7
support count by hour/day/month…
videos_view_count
video_id BIGINT K
ts TIMESTAMP
view_count COUNTER ++

Refining


Q1
Q2
Q4
Q5
Q6
videos_by_releasedate
release_year INT K
release_date TIMESTAMP
video_id BIGINT
title TEXT
videos_by_genre
genre TEXT K
release_date TIMESTAMP
video_id BIGINT
title TEXT
views_by_user
title TEXT
views_by_country_day
video_id BIGINT
user_id BIGINT
views_by_video
o_id BIGINT K
_id TIMEUUID
ce TEXT
_id BIGINT
Reminder: counter type can not be used with other types in the same table (except for the primary key)
Q4
Q3
videos_by_id
video_id BIGINT K
release_date TIMESTAMP
title TEXT
{genres} SET<TEXT>
Q3
Q7
support count by hour/day/month…
videos_view_count
video_id BIGINT K
ts TIMESTAMP
view_count COUNTER ++

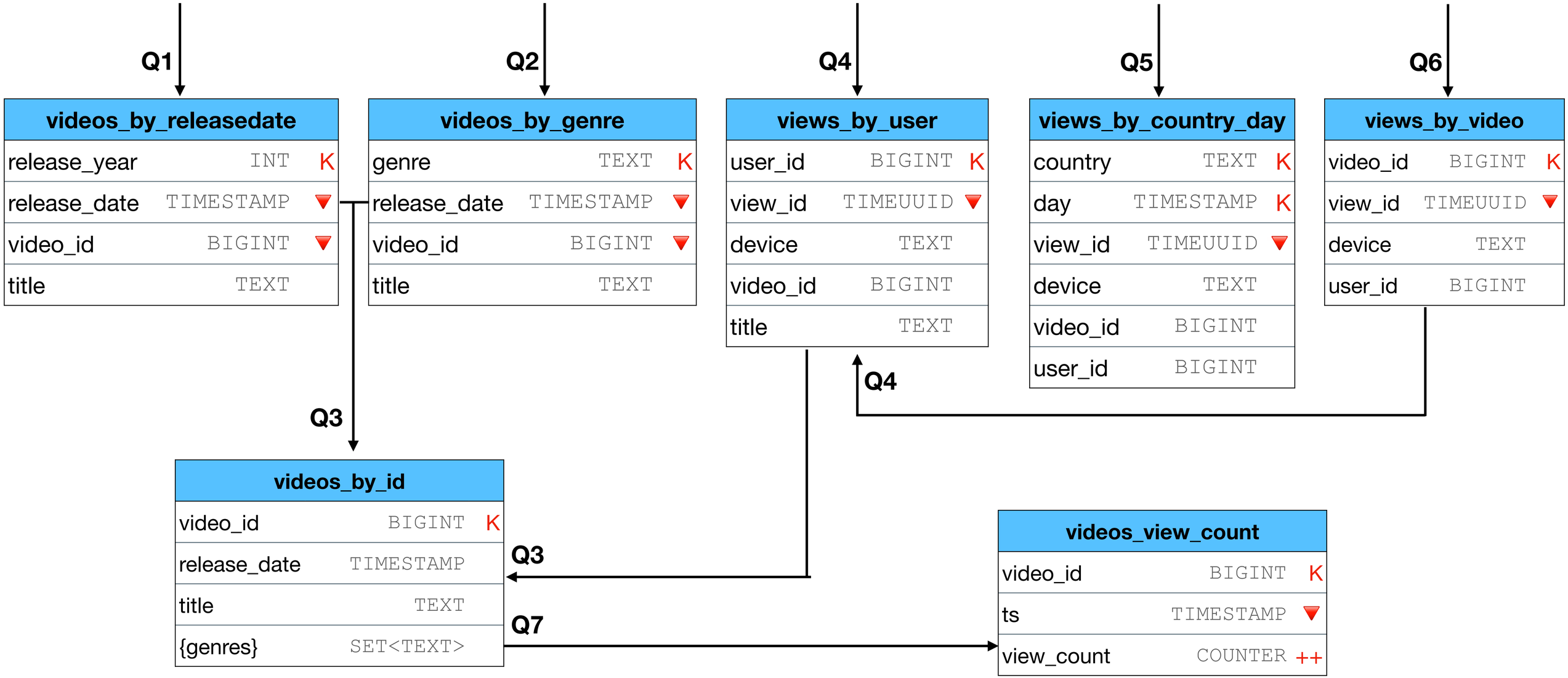
Evaluating and optimizing - example (2)

- The streaming service is a big hit
  - More users
  - More usage

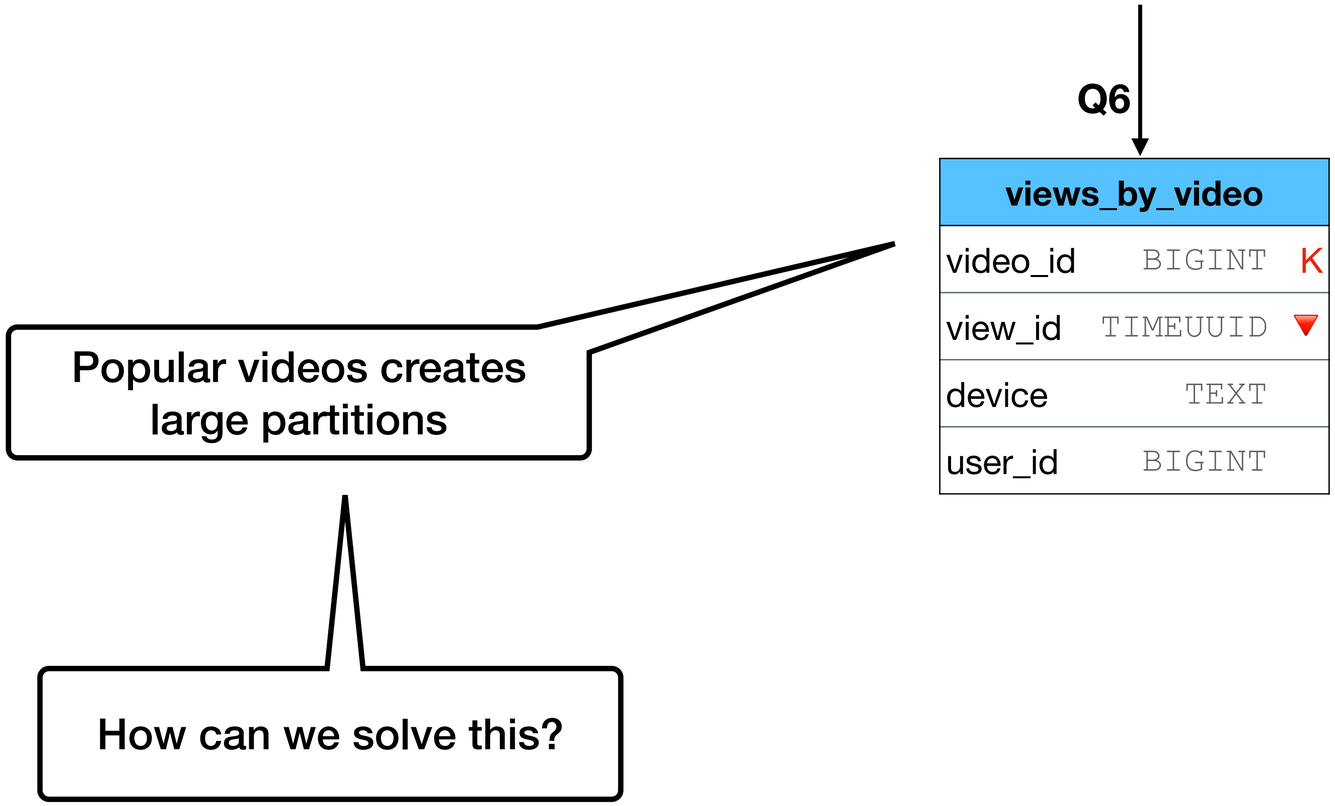
Suddenly

- Queries are getting slower
- Large partitions warnings on the logs
- **Adding more Cassandra servers does not help**

Do you see a problem?



Do you see a problem?



Do you see a problem?

See you next week :)

Popular videos creates large partitions

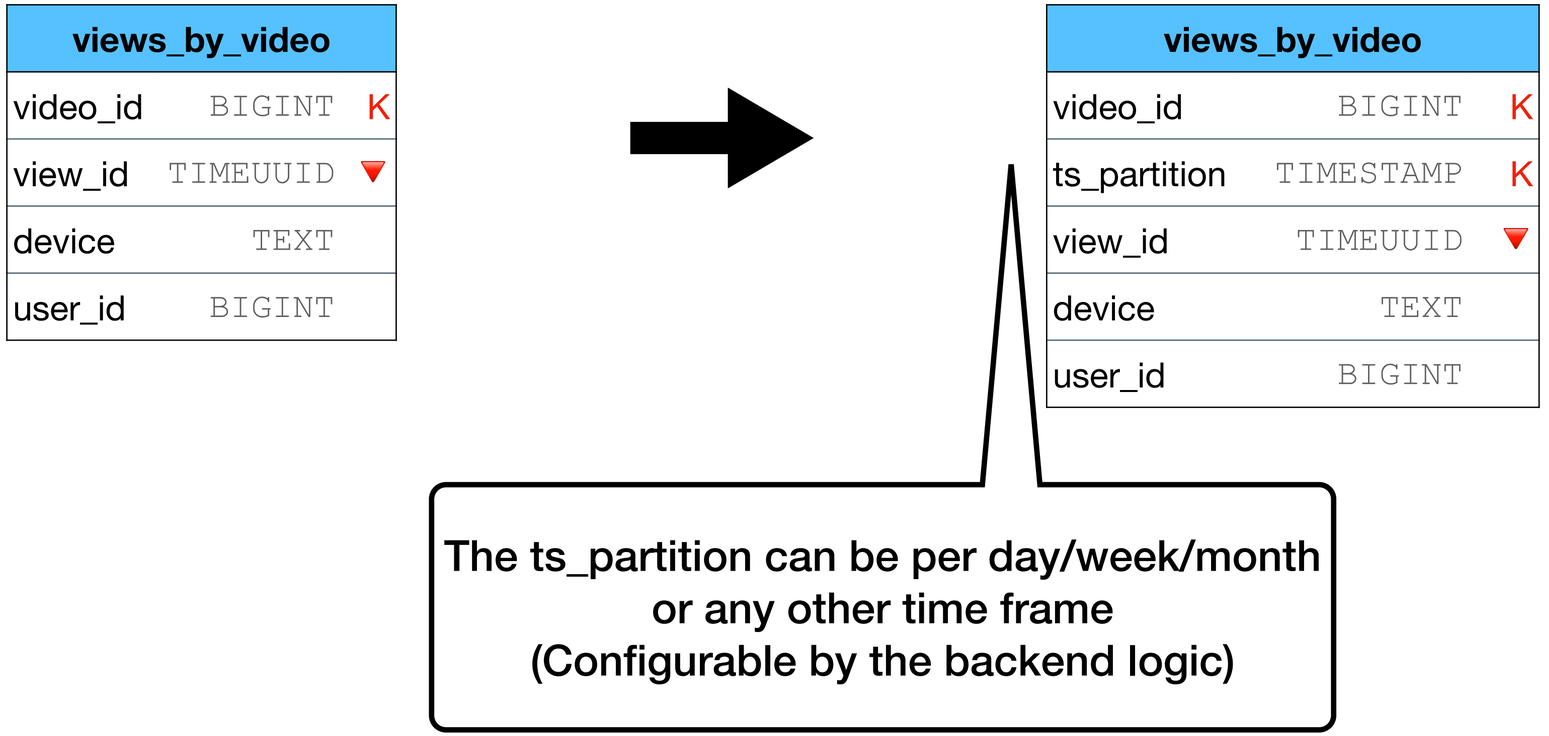
How can we solve this?

Q6

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

Altering the schema

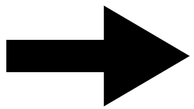
- We need to partition the data differently



Altering the schema

- We need to partition the data differently

views_by_video		
video_id	BIGINT	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	



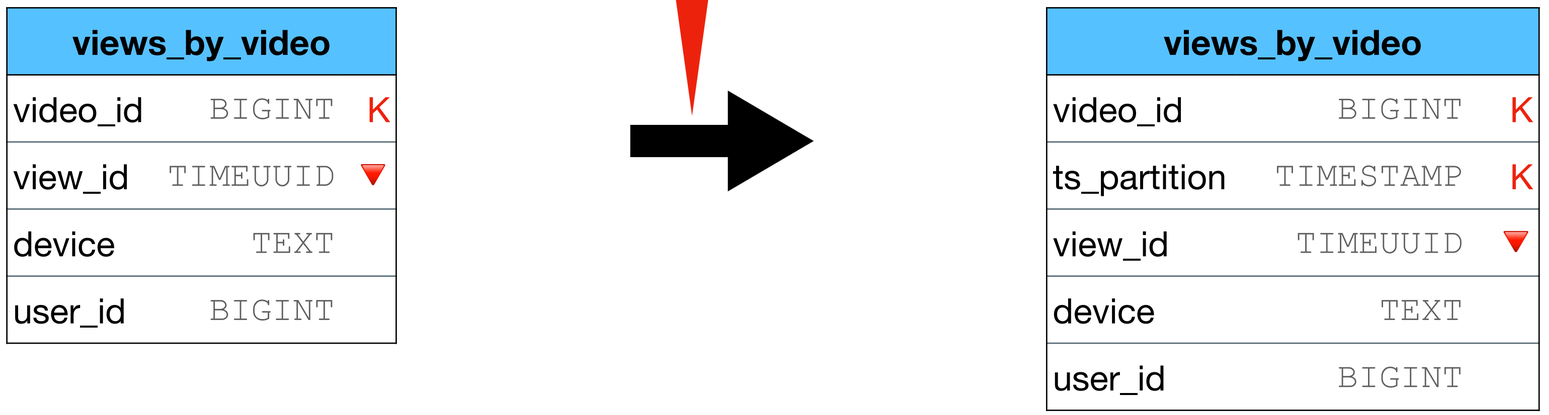
views_by_video		
video_id	BIGINT	K
ts_partition	TIMESTAMP	K
view_id	TIMEUUID	▼
device	TEXT	
user_id	BIGINT	

- We will need to issue more than 1 query to retrieve the data
  how much?
- Not an issue as this query is done during a model build of the recommendation engine and not in real time

Alte

Note - this might not be the optimal solution.
We will talk about more ways to partition the data soon

- We



- We will need to issue more than 1 query to retrieve the data
  how much?
- Not an issue as this query is done during a model build of the recommendation engine and not in real time

Fixing the tracks of a moving train

- The table cannot be altered - a new one is needed
- You service works 24x7, you cannot stop it
- An “online merge” is required

- **Not a trivial update**
  Happens all the time for growing products

More “popular problems”

- Large partitions
- Application logic changes
  new entities, new queries
- Imbalanced data
- Unforeseen hotspots