Relational Modeling

Big Data Systems

Dr. Rubi Boim

Motivation (for this course)

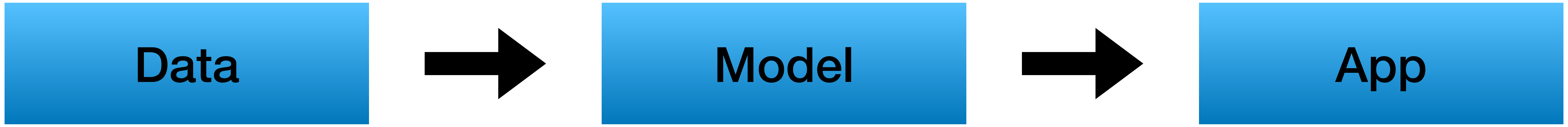
- Data modeling is an important process when creating a relational database

- Data modeling is **the most important** process when creating a **big data database**

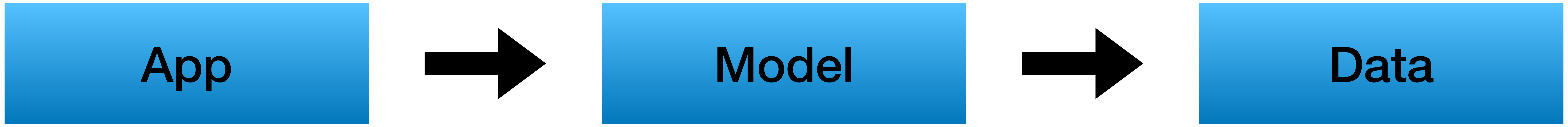
- Modeling for NoSQL is “different” than relational
  understanding relational modeling in crucial for wide column modeling

Relational vs NoSQL - design

- Relational
  focus on entities



- NoSQL
  focus on queries



Relational data modeling

Modeling is an Art

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

Relational Modeling - general steps

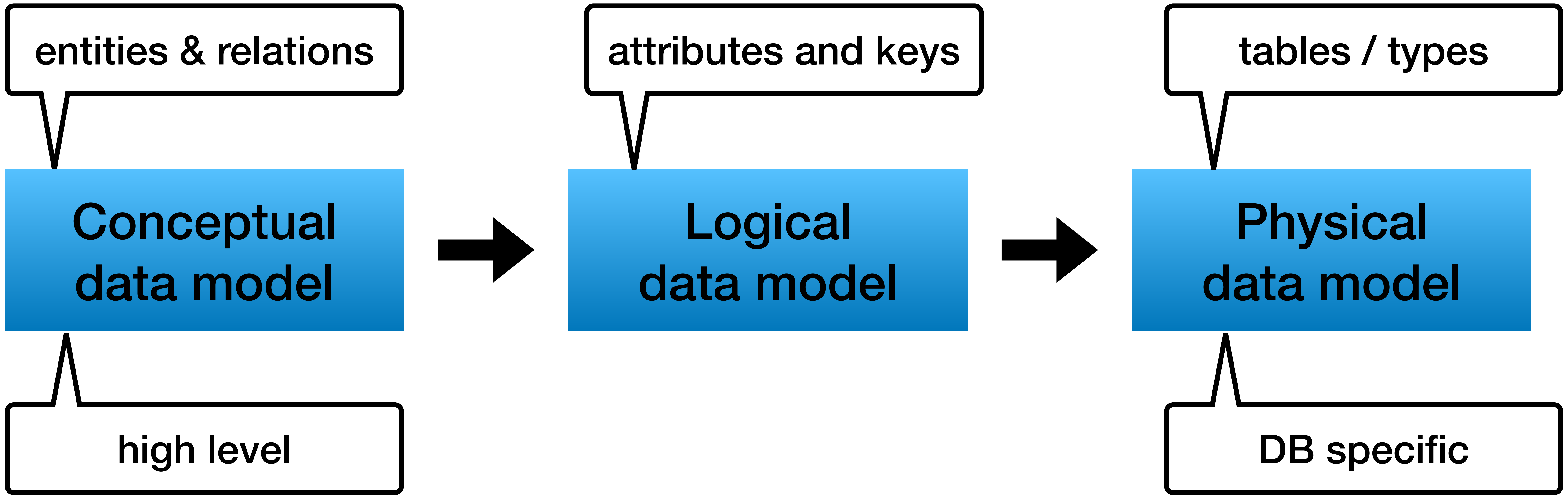
- Map conceptual entities, attributes and their relations

- Map primary and foreign keys

- Define data types

- Create tables

Relational Modeling - 10,000 foot view



entities & relations

Conceptual data model

high level

attributes and keys

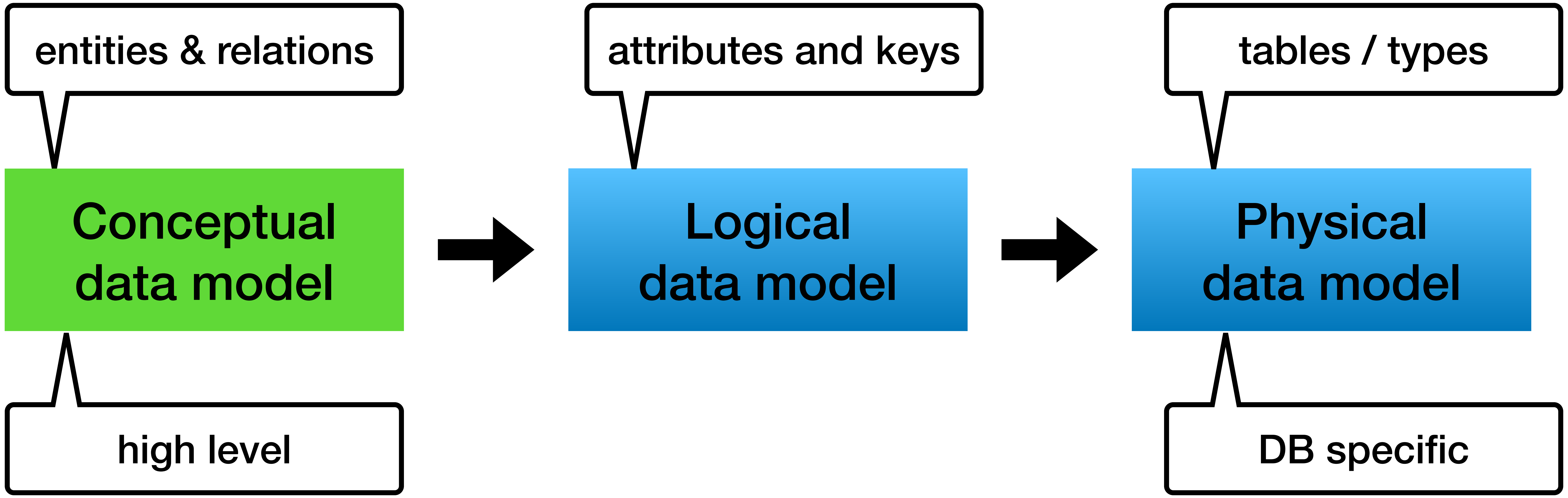
Logical data model

tables / types

Physical data model

DB specific

Relational Modeling - 10,000 foot view



entities & relations

Conceptual data model

high level

attributes and keys

Logical data model

tables / types

Physical data model

DB specific

Conceptual data model

- Abstract view of the world

  server and database types are irrelevant

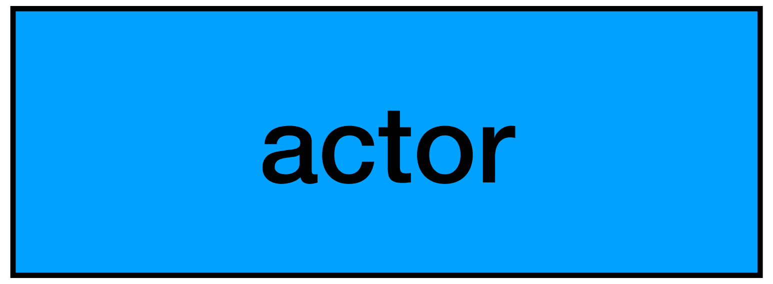
- Can be defined by non technical teams

  not really in reality…

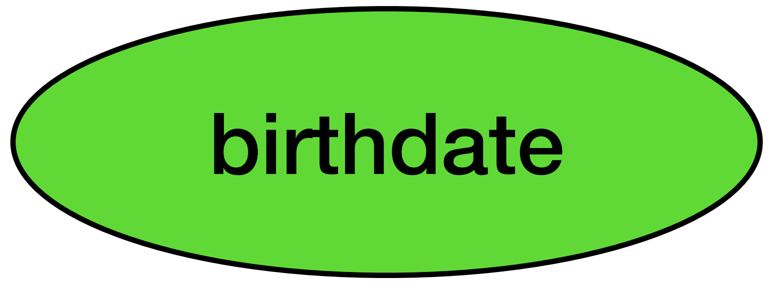
- Entity / Relationship model (ER)

ER Model

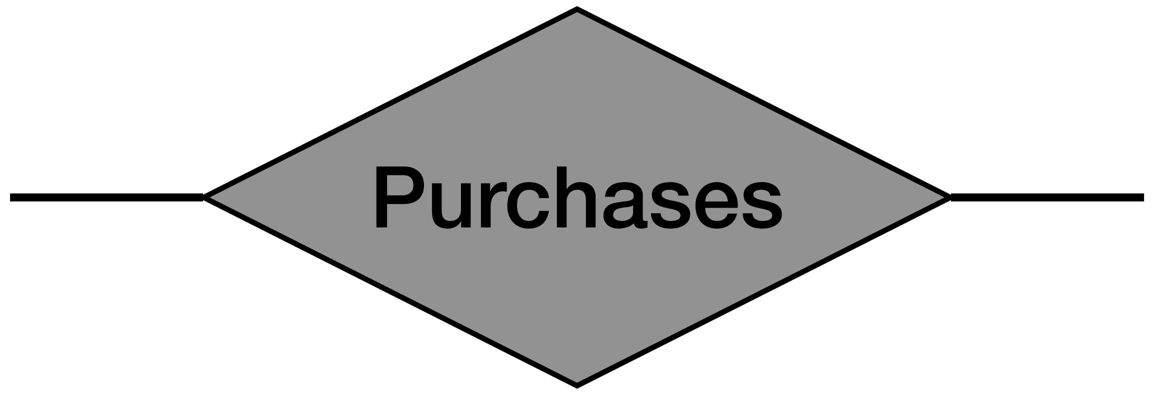
- Entities



- Attributes



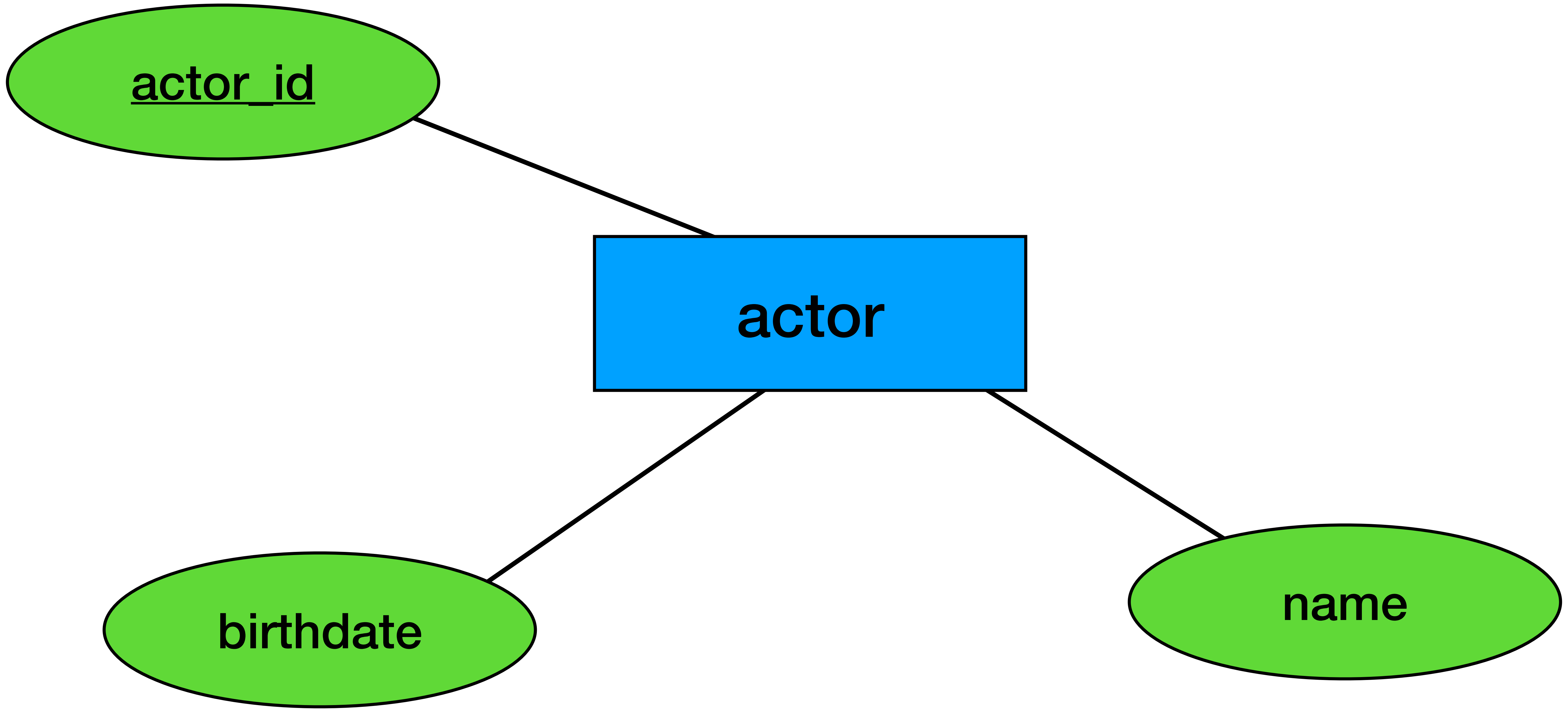
- Relations
  between entities



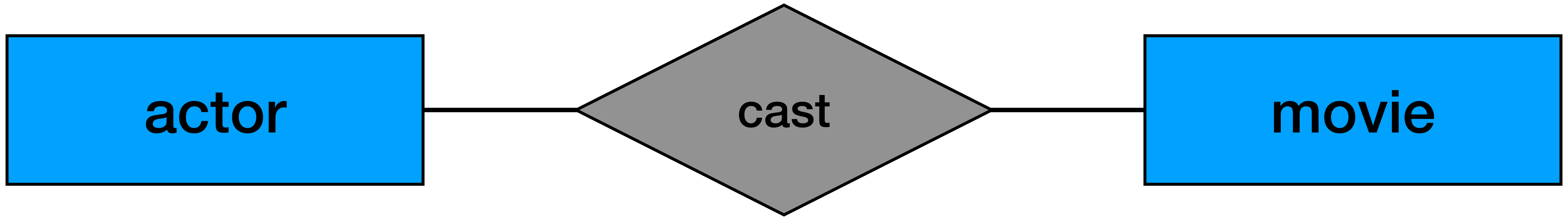
* There are more types like ISA (is a)

Entity

- Each entity must have a key

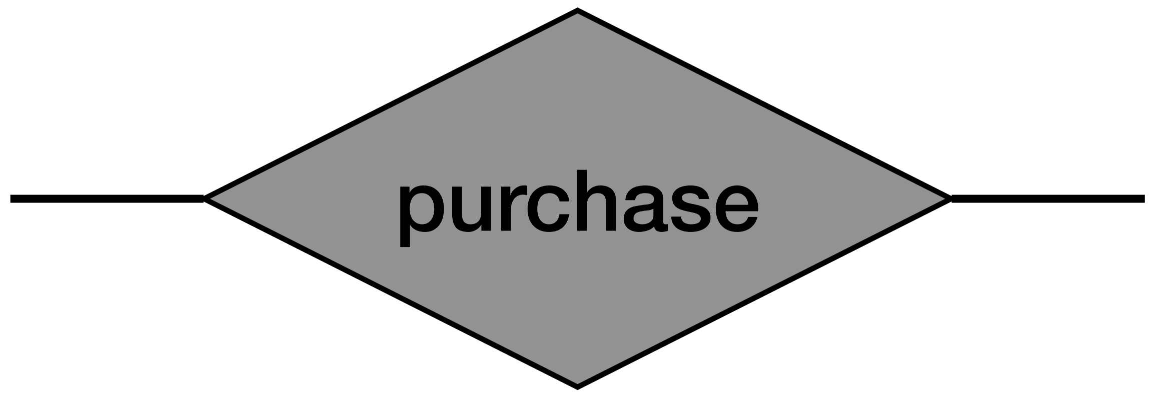


Relation (between entities)

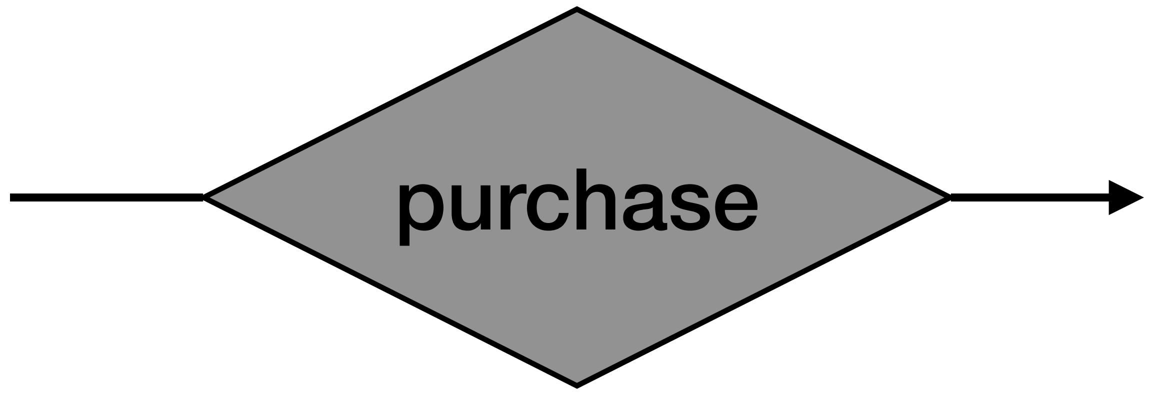


Cardinality (of relation)

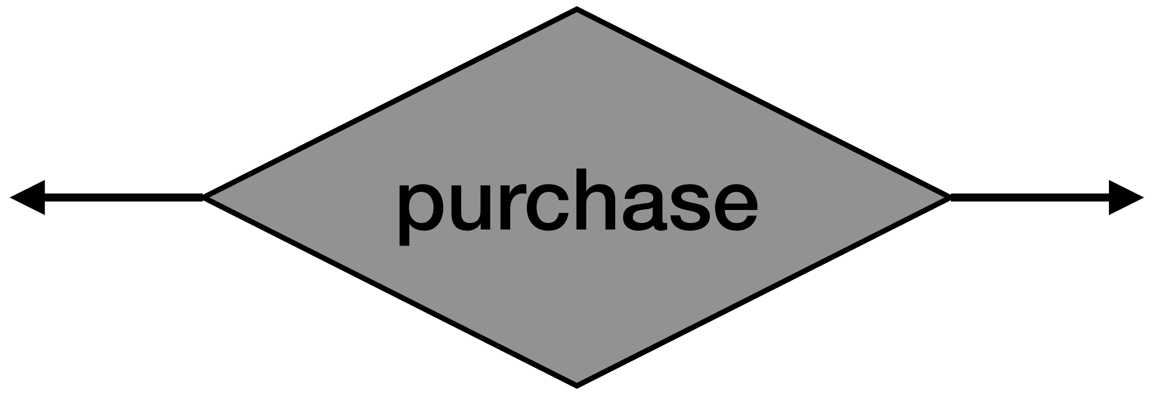
- cardinality is the number of occurrences in one entity which are associated to the number of occurrences in another



many-many

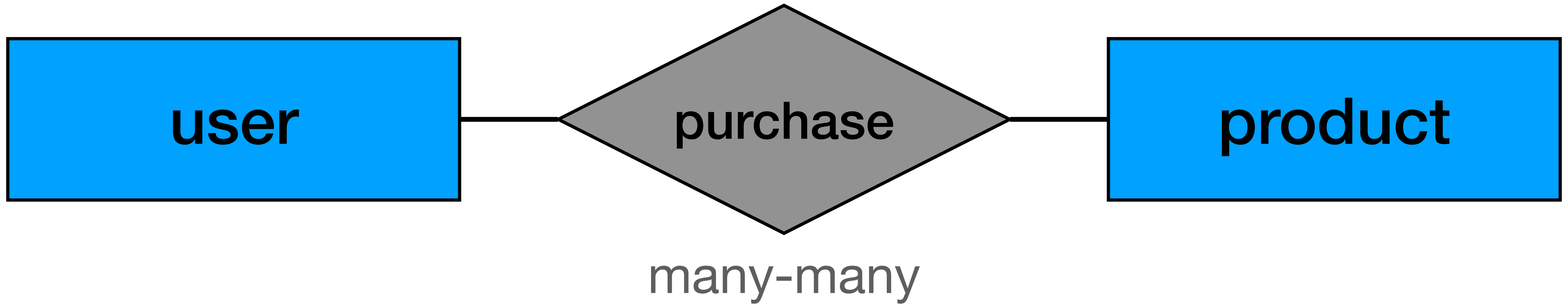


many-one

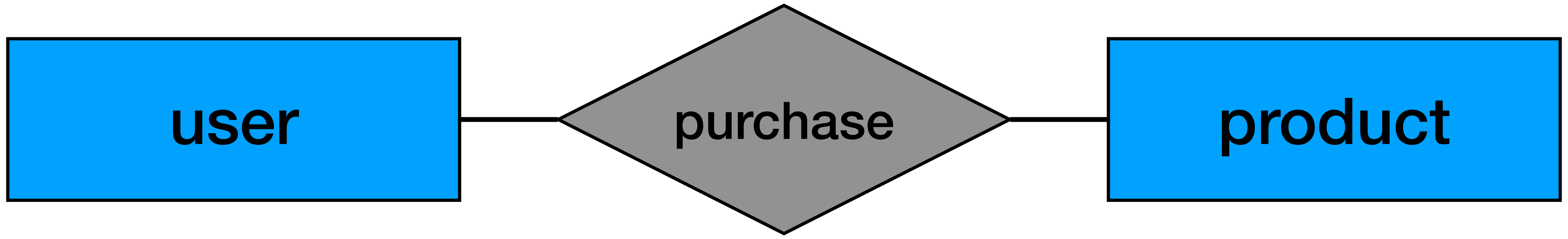


one-one

Many to Many



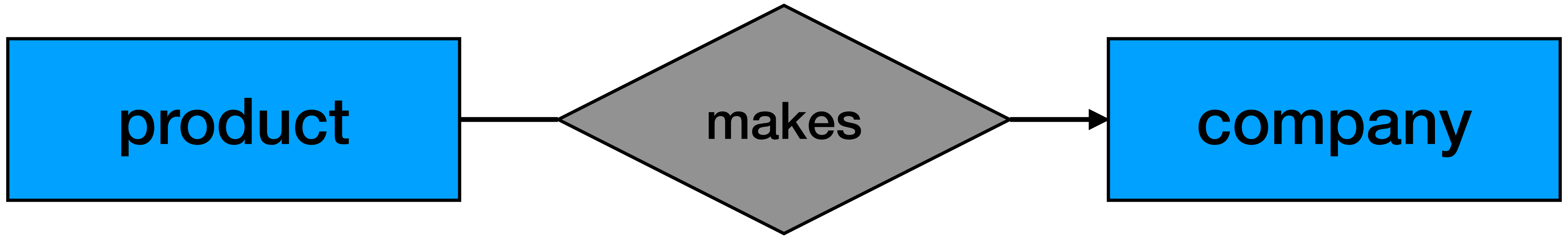
Many to Many



many-many

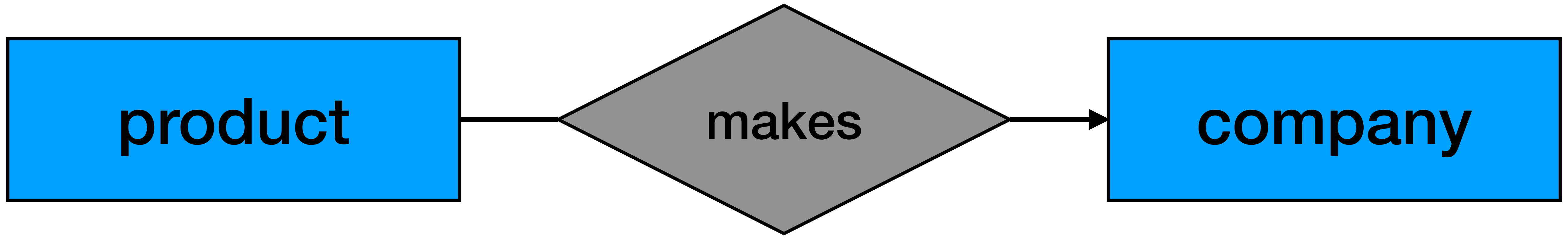
Each user can buy many products
(but each product only once)

Many to One



many-one

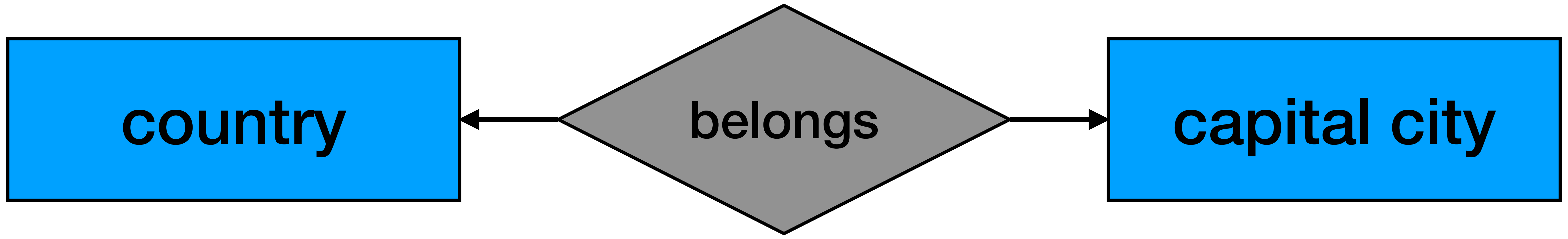
Many to One



many-one

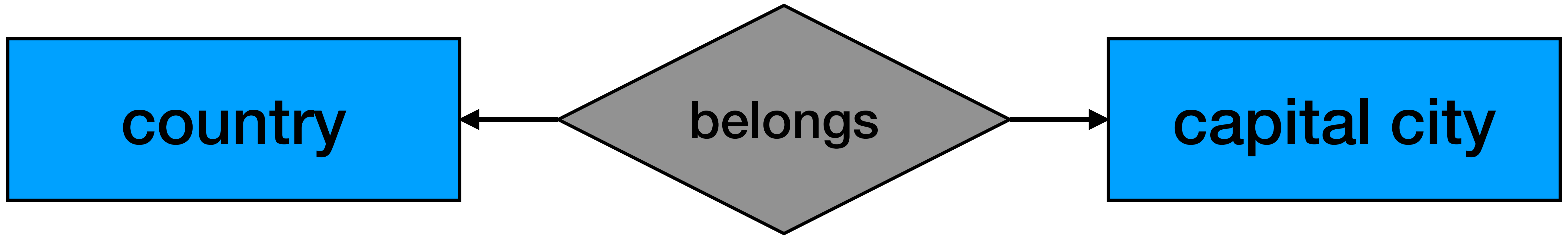
Each product is made by one company

One to One



one-one

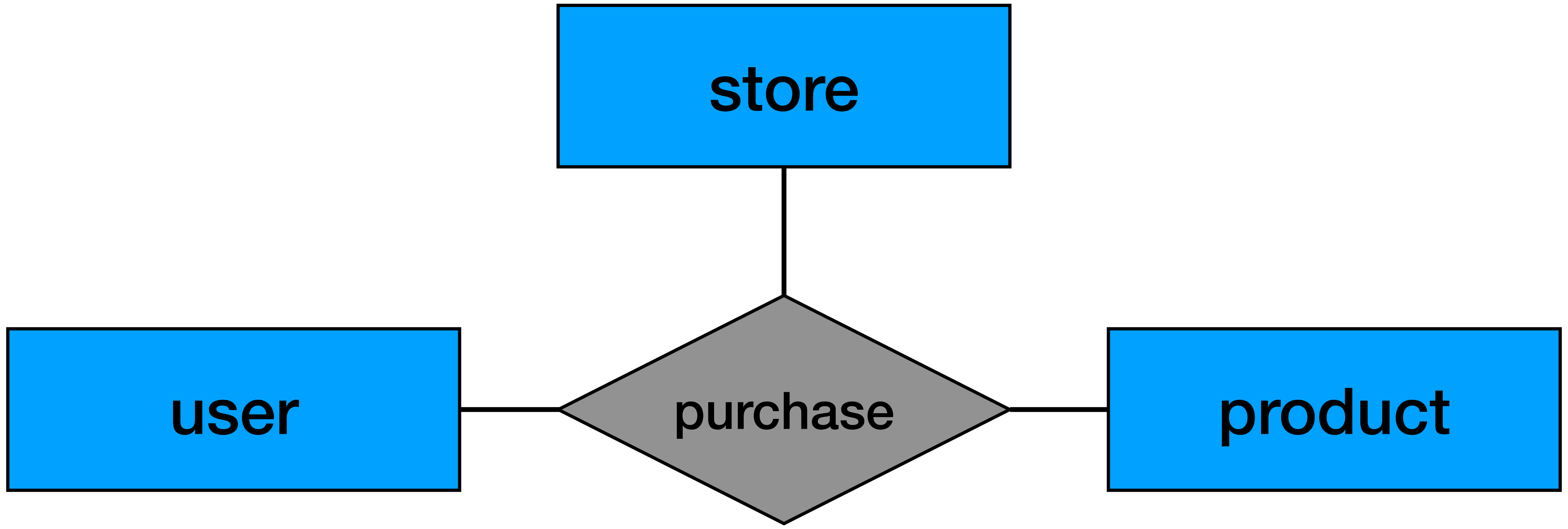
One to One



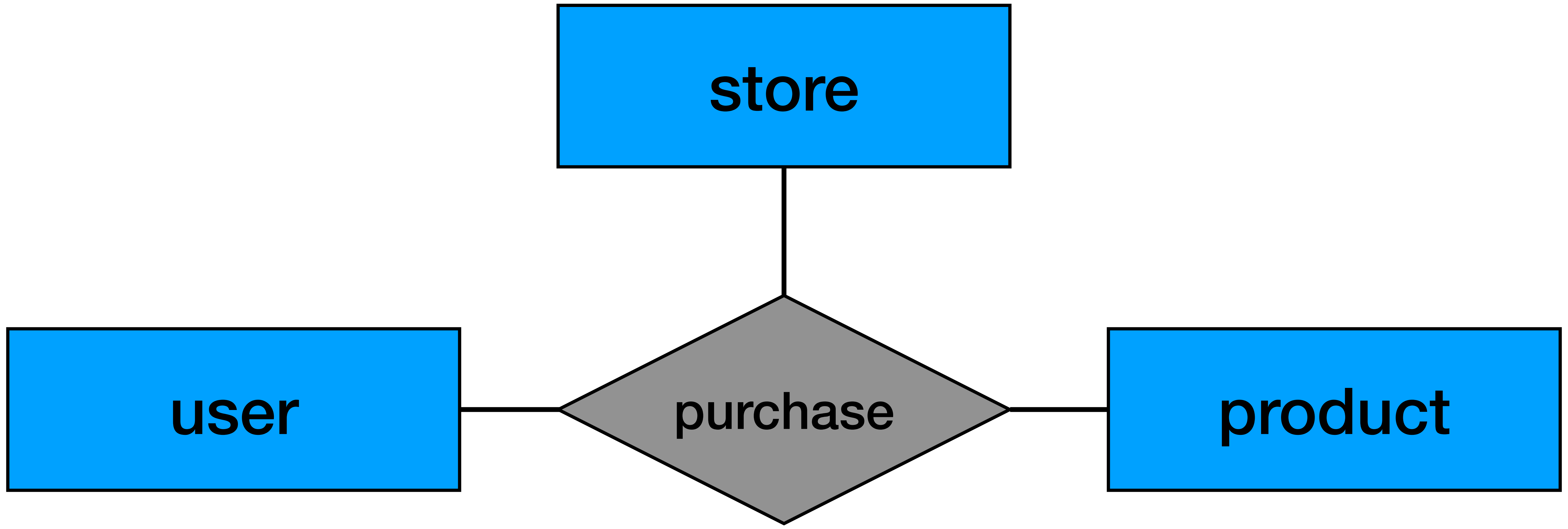
one-one

Each country has one capital city, and each capital city belongs to one country

Multi way relations

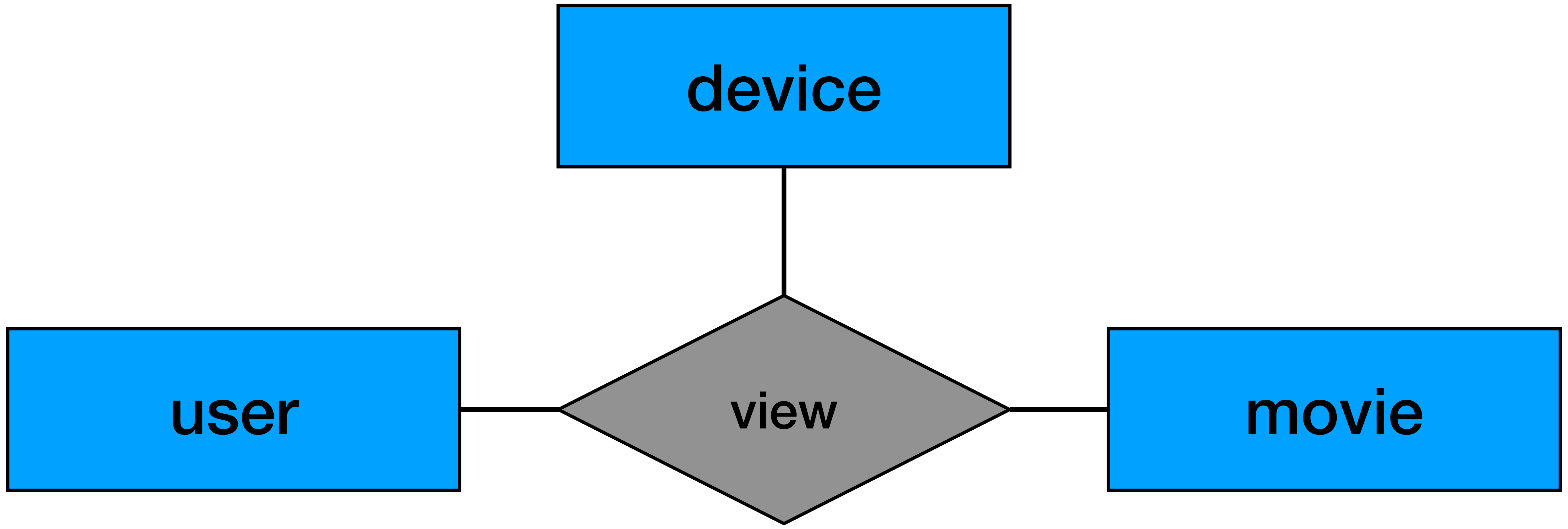


Multi way relations

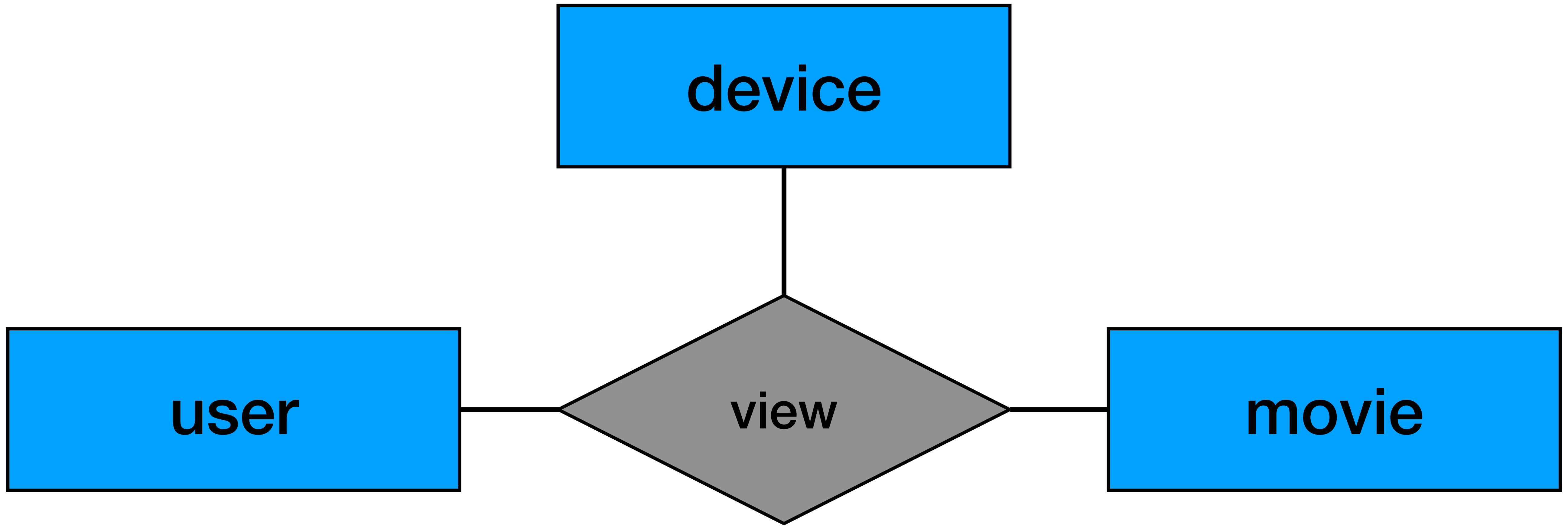


Each user can buy many products in different stores (but user-store-product combination only once)

Multi way relations (another example)

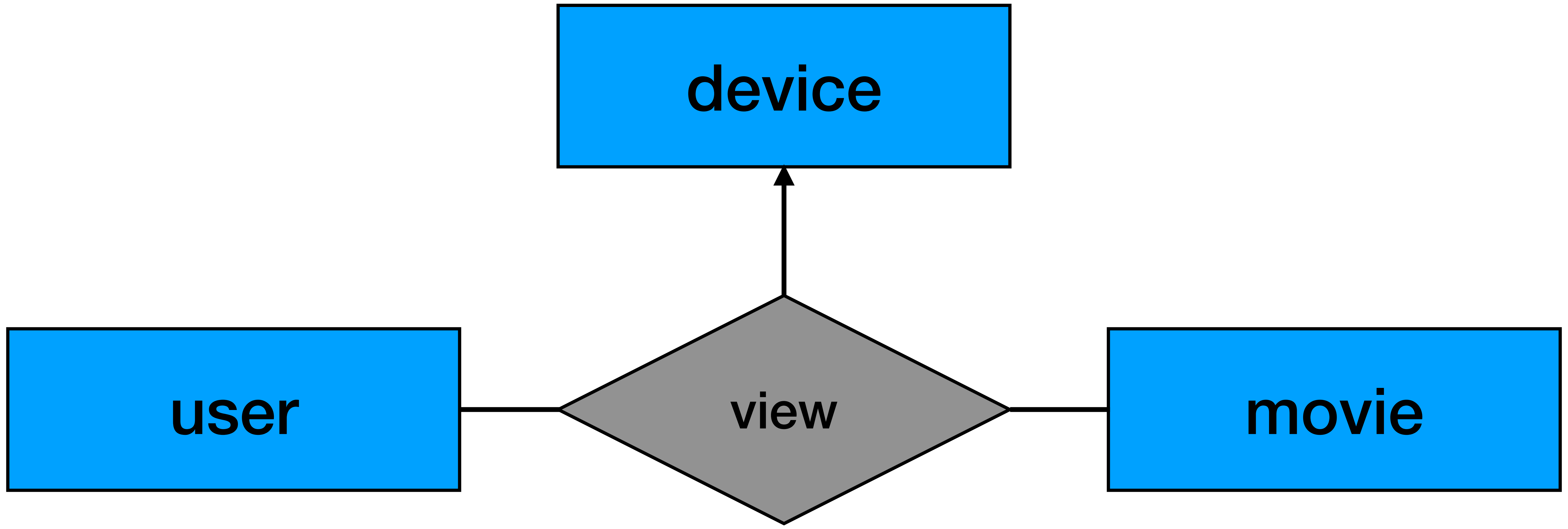


Multi way relations (another example)

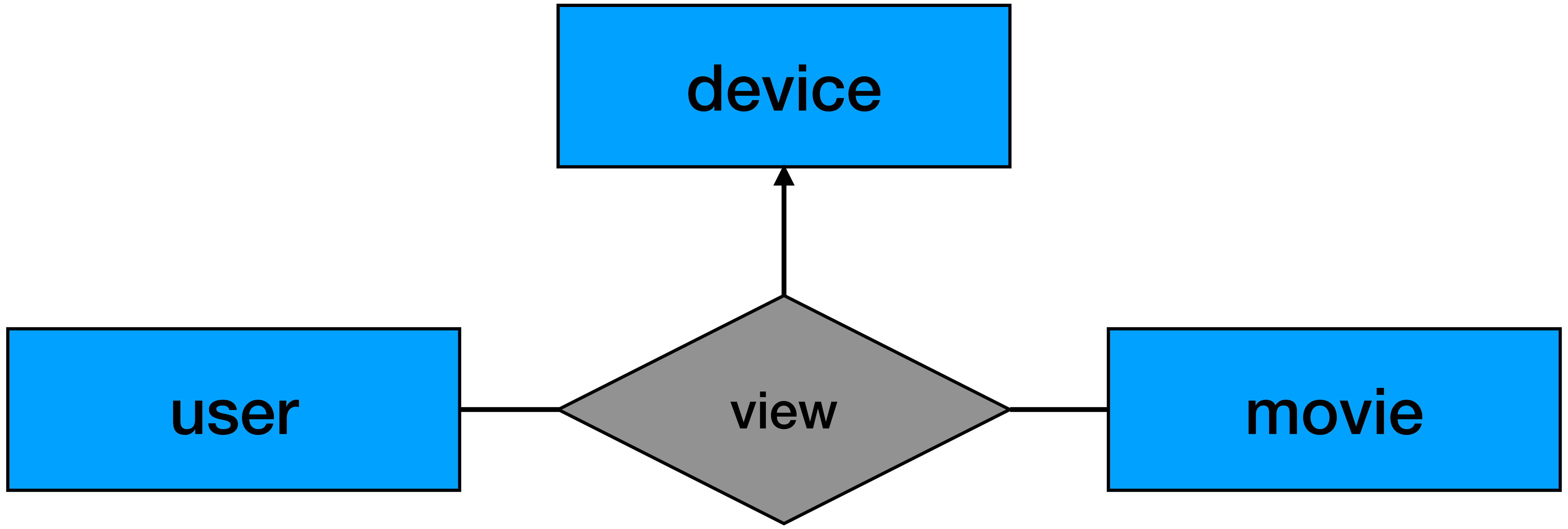


Each user can view many movies on different devices (but user-movie-device combination only once)

Multi way relations + cardinality

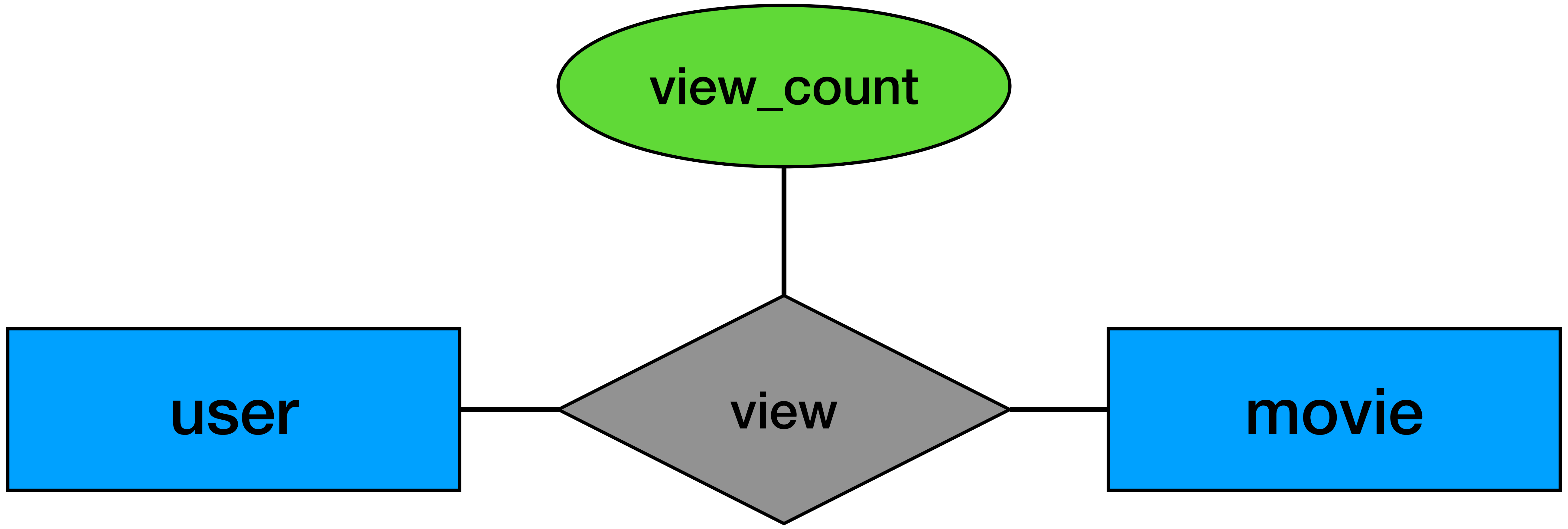


Multi way relations + cardinality

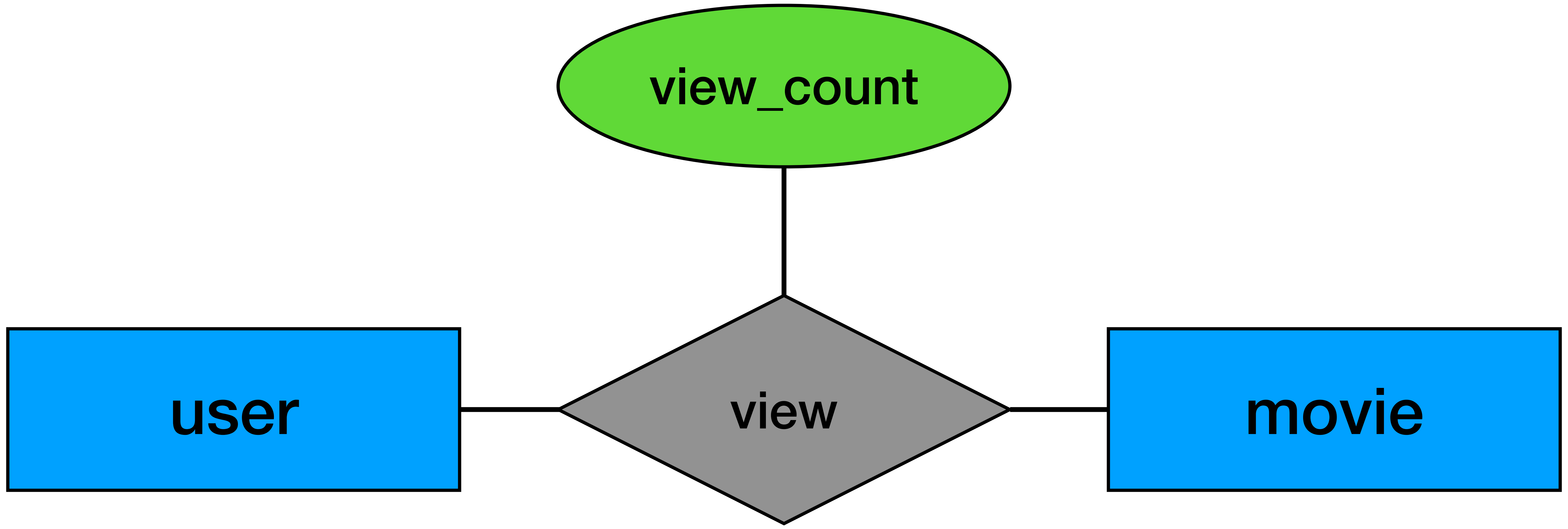


Each user can view many movies.
If we know the user and the movie, we know the device

Attributes for relations

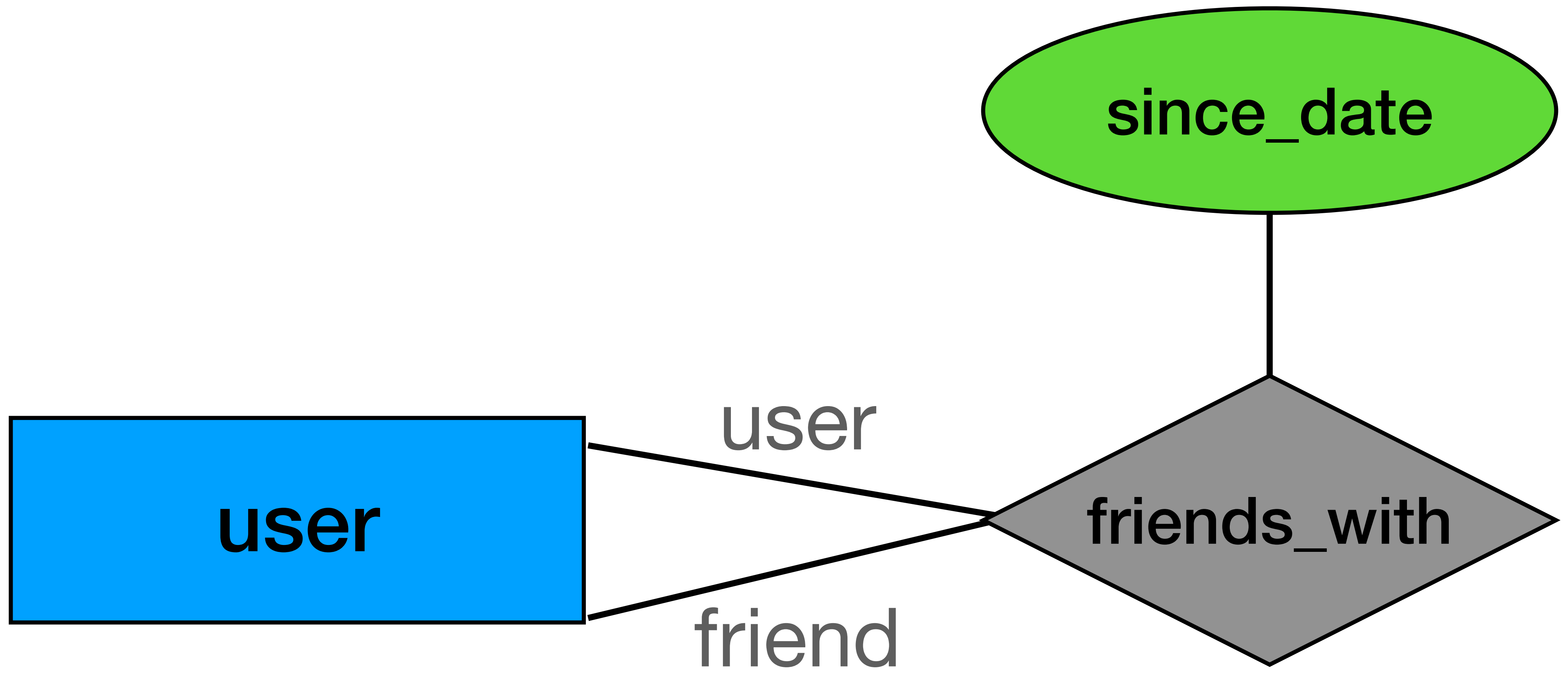


Attributes for relations

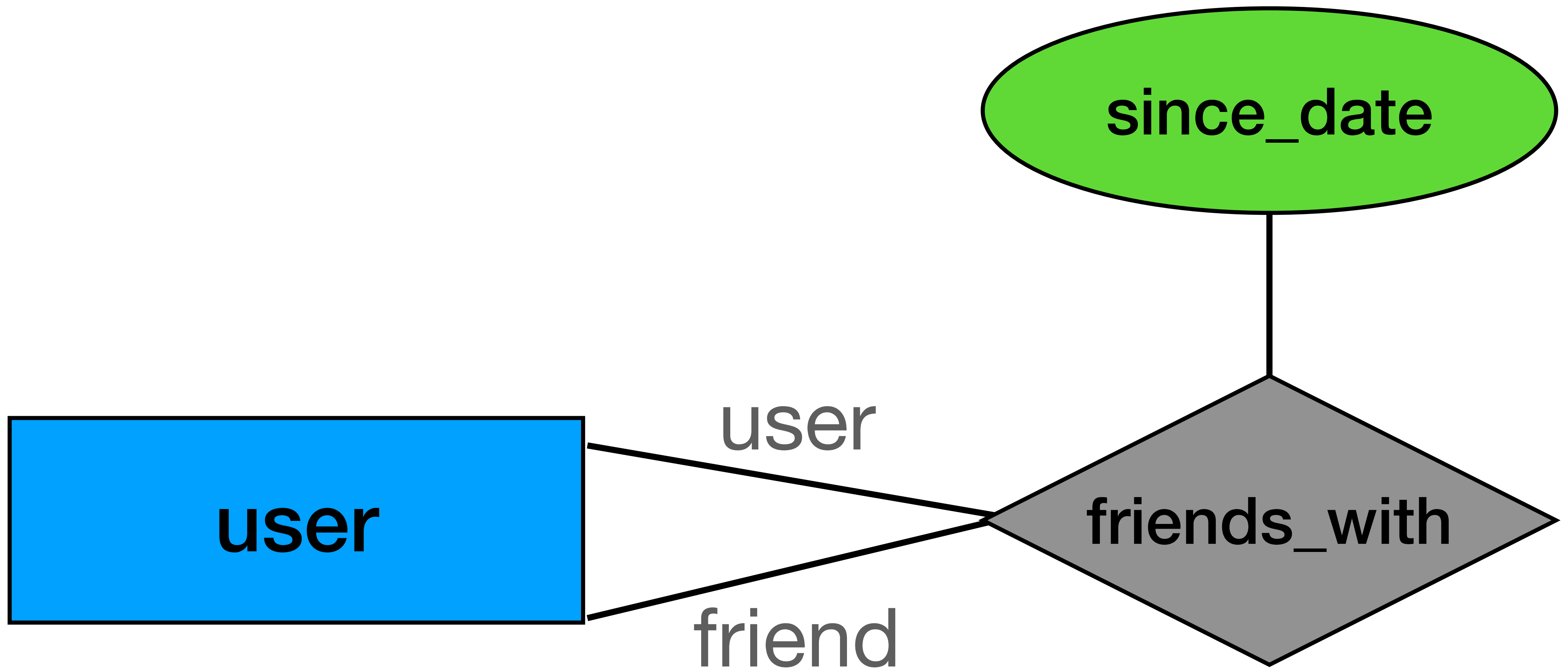


Each user can view many movies.
For each “view” we also save the view_count

Roles in relations



Roles in relations



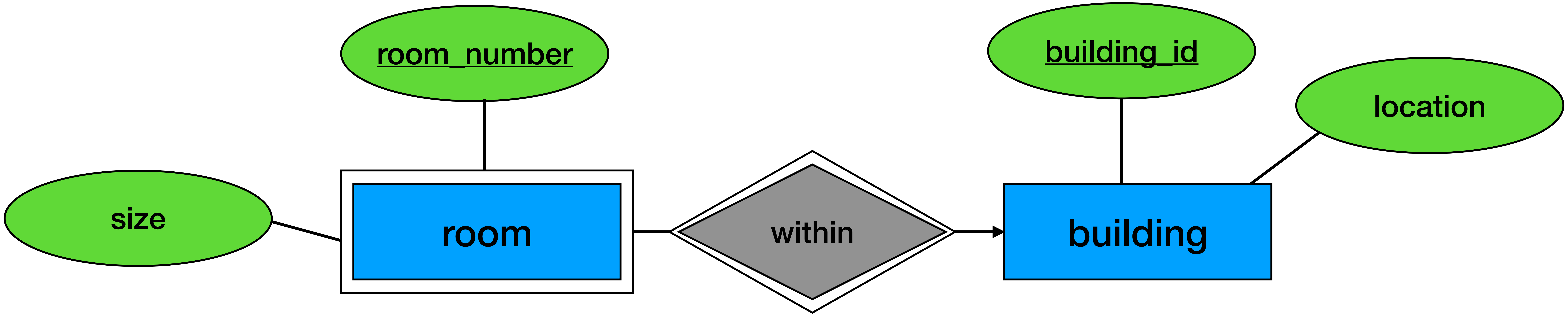
A user can be friends a different user

From previous class:

`friends(user_id, friend_user_id, since_date)`

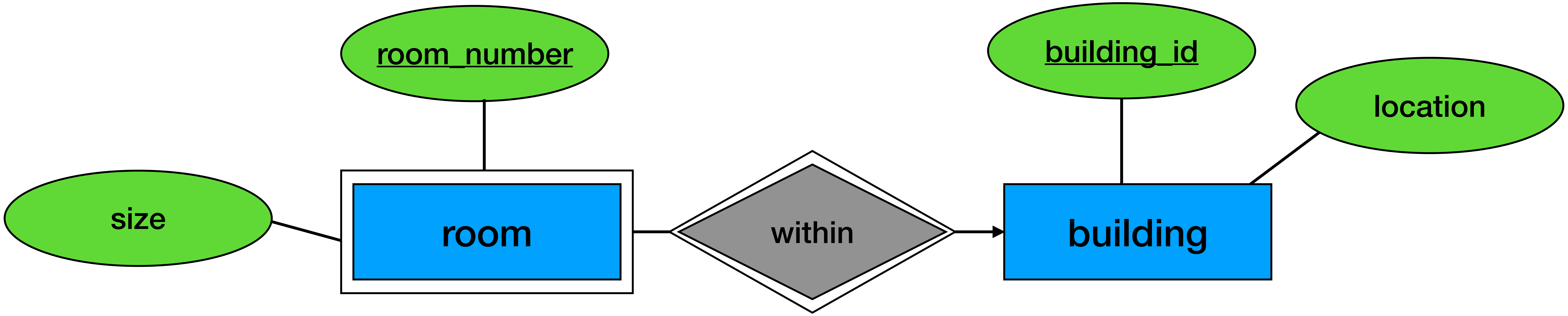
Weak Entity

- When some of their keys comes from other entities



Weak Entity

- When some of their keys comes from other entities



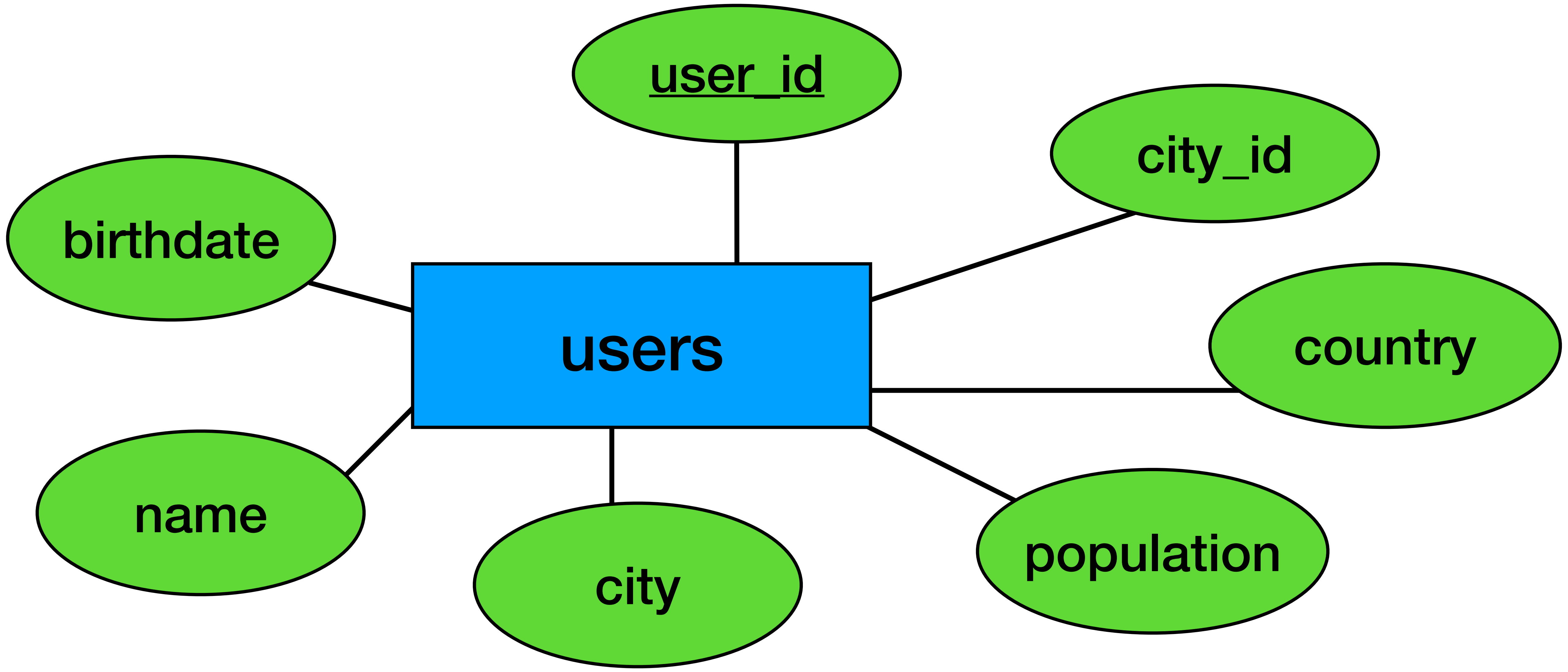
In this example, the key for room is building_id and room_number

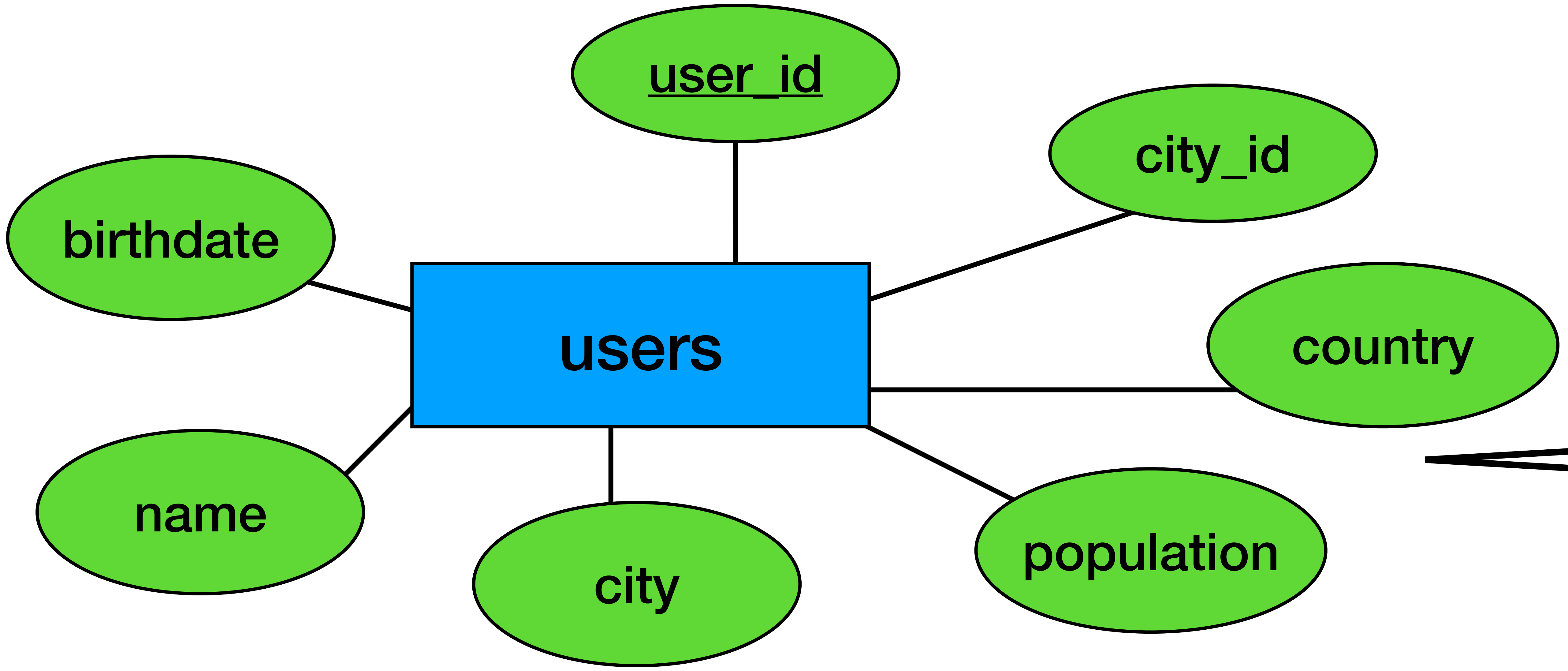
Example

Story time

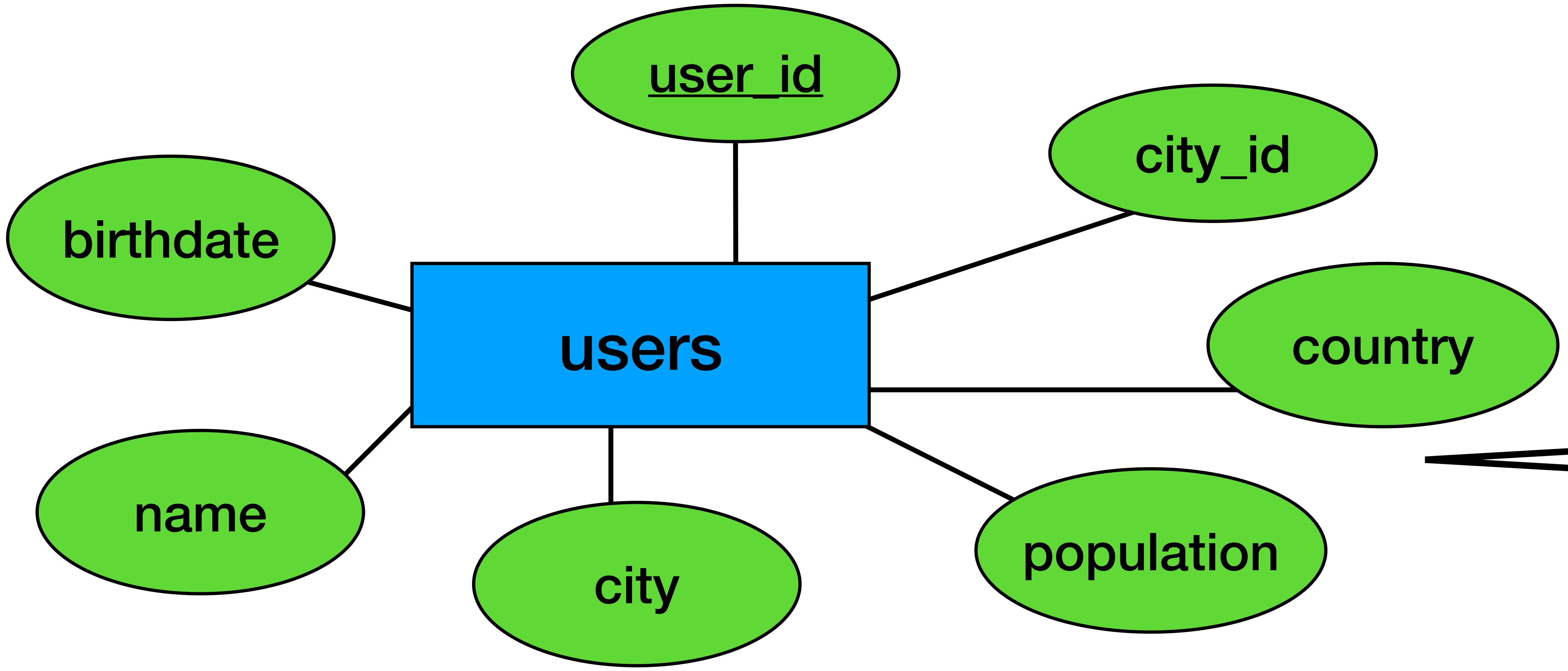
Design an ER diagram for a video platform:

- A user is defined by user_id. We also save her name, birthdate and city. For each city we save the city_id, name, population and country
- A video is defined by a video_id and we store its genre, release date and title
- For each video we keep the actors that appears in it along with their character name.
- The actors are defined by an actor_id along with their name
- For analytics, if a user views a video we save the most recent viewing timestamp


users
user_id
city_id
birthdate
country
name
city
population



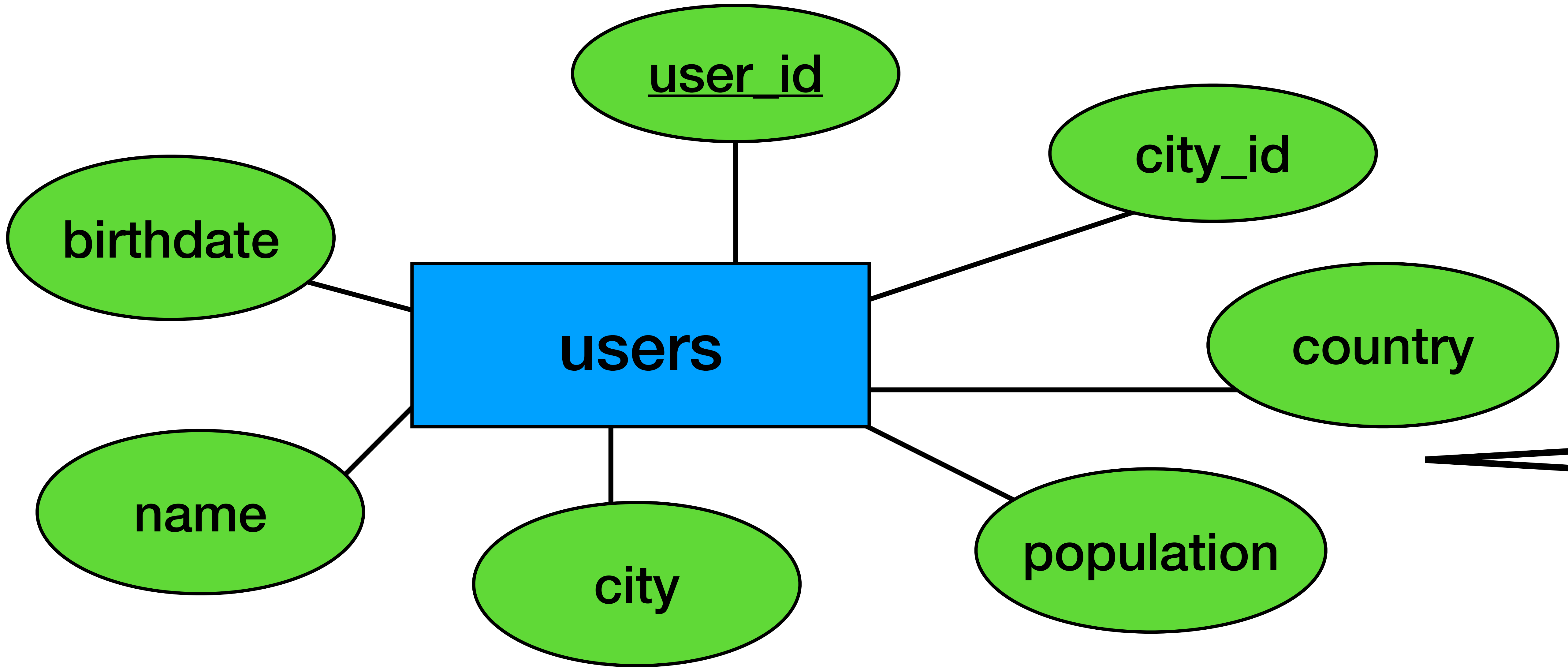
Do we save duplicate data?



Do we save duplicate data?

Intuition

“If we have duplicate data
—>
The database is NOT normalized”
(3NF / BCNF)

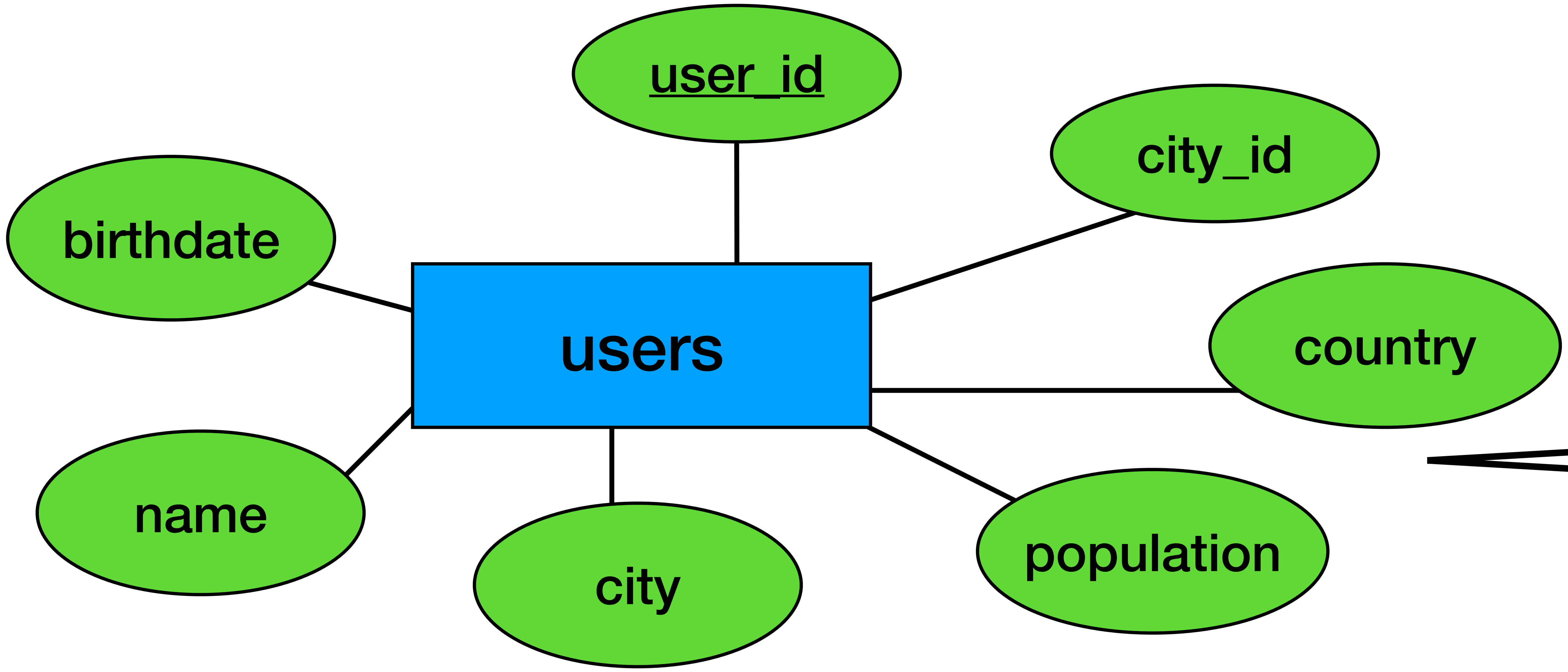


Do we save duplicate data?

Intuition

“If we have duplicate data
—>
The database is NOT normalized”
(3NF / BCNF)

In Relational Database we want the model to be normalized



Do we save duplicate data?

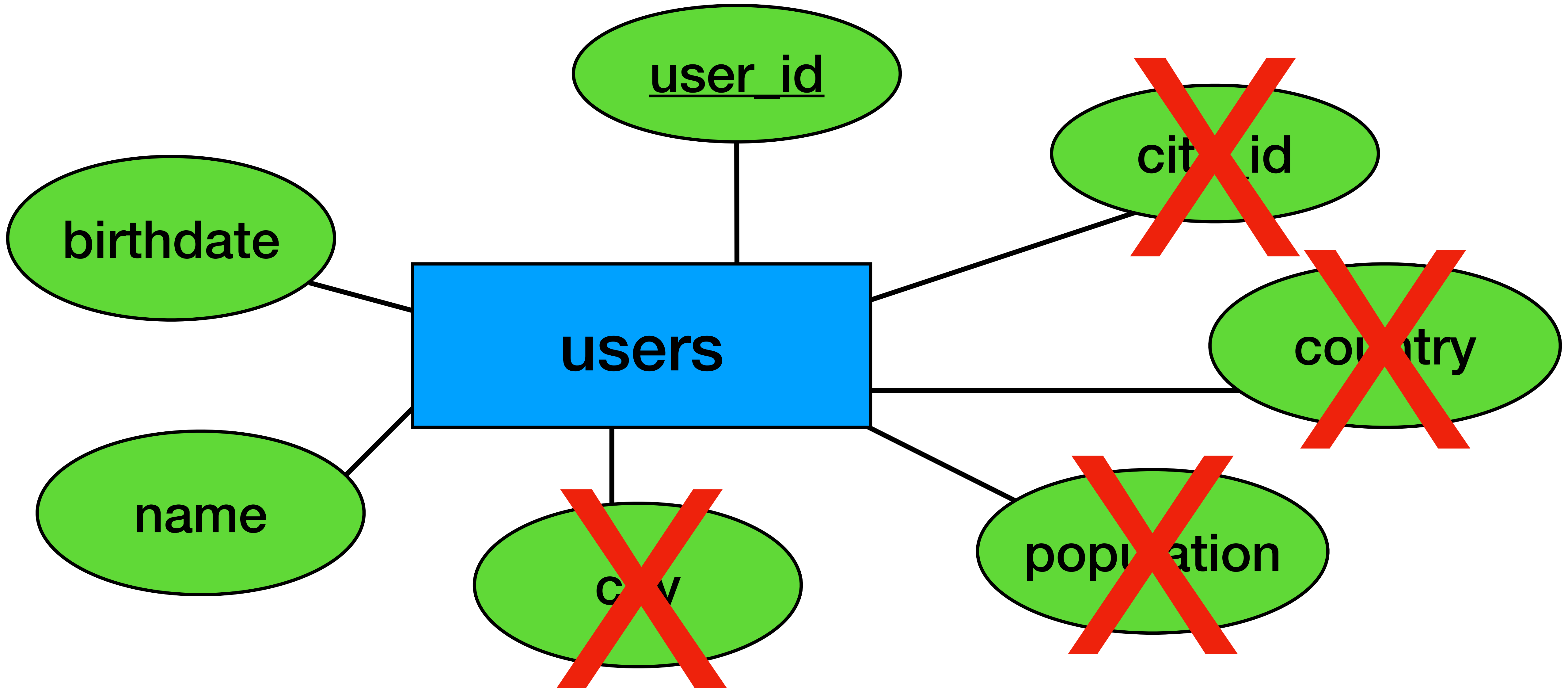
Intuition

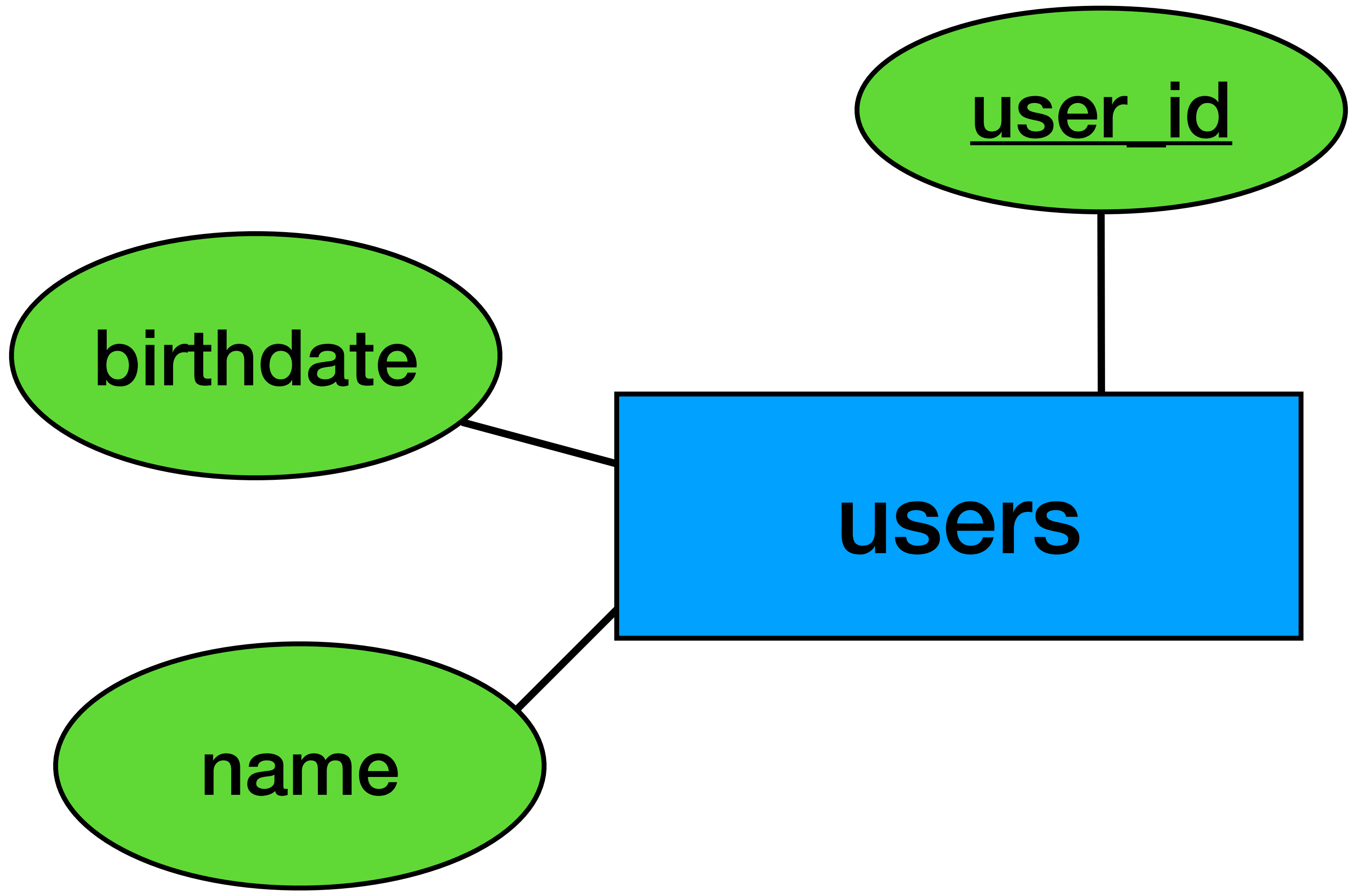
“If we have duplicate data
—>
The database is NOT normalized”
(3NF / BCNF)

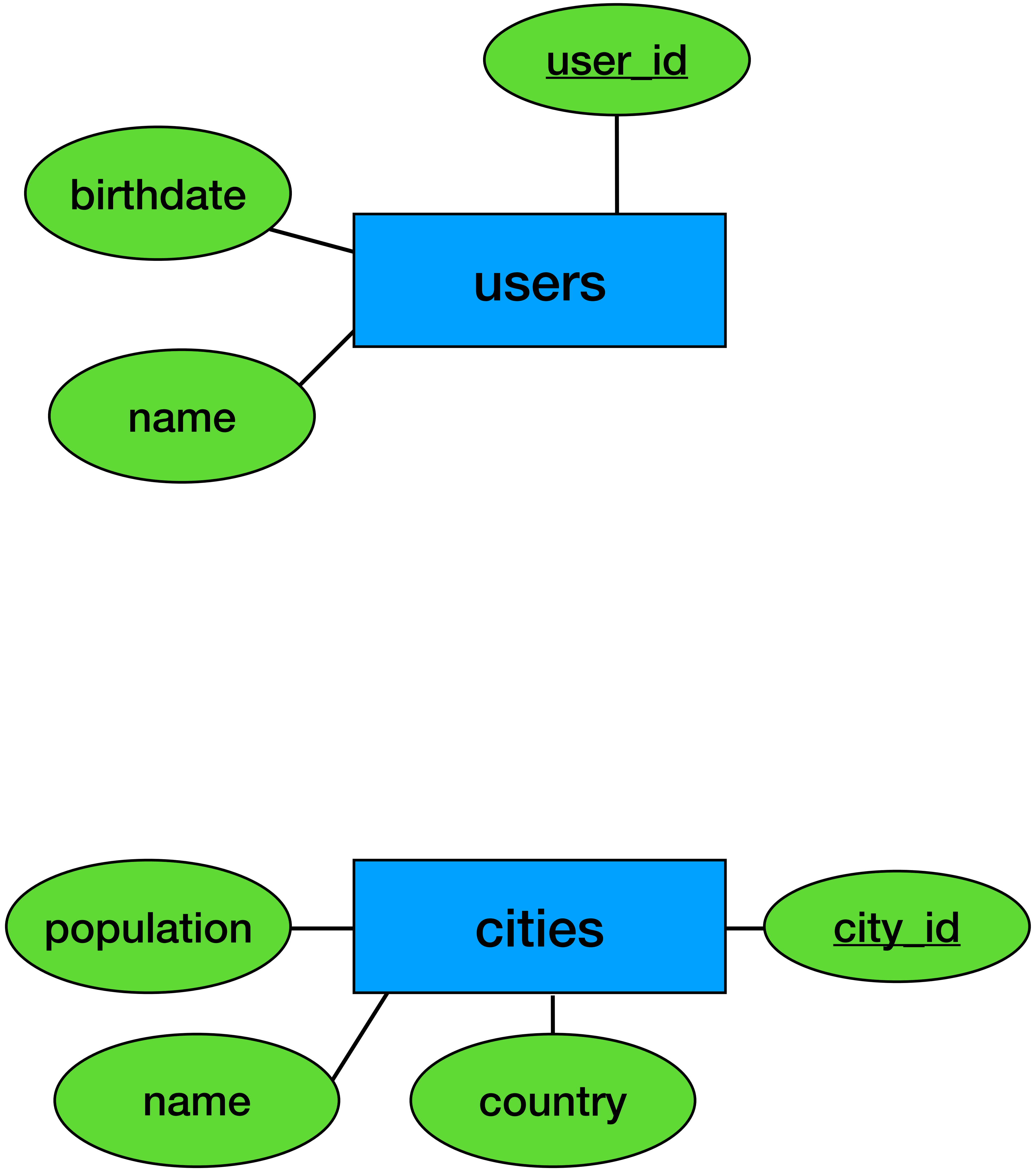
In Relational Database we want the model to be normalized

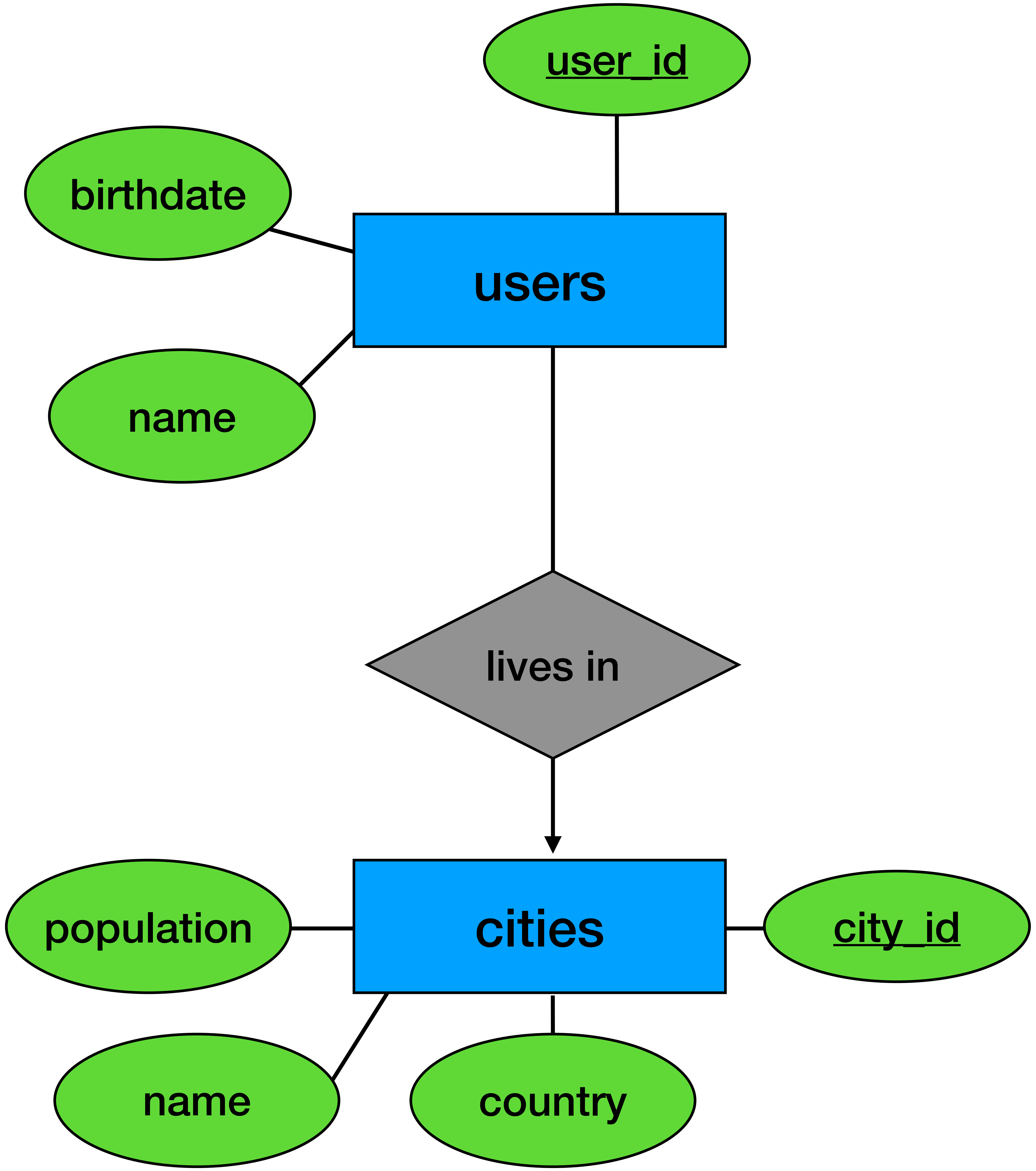
This is NOT the case in Wide Column DBs

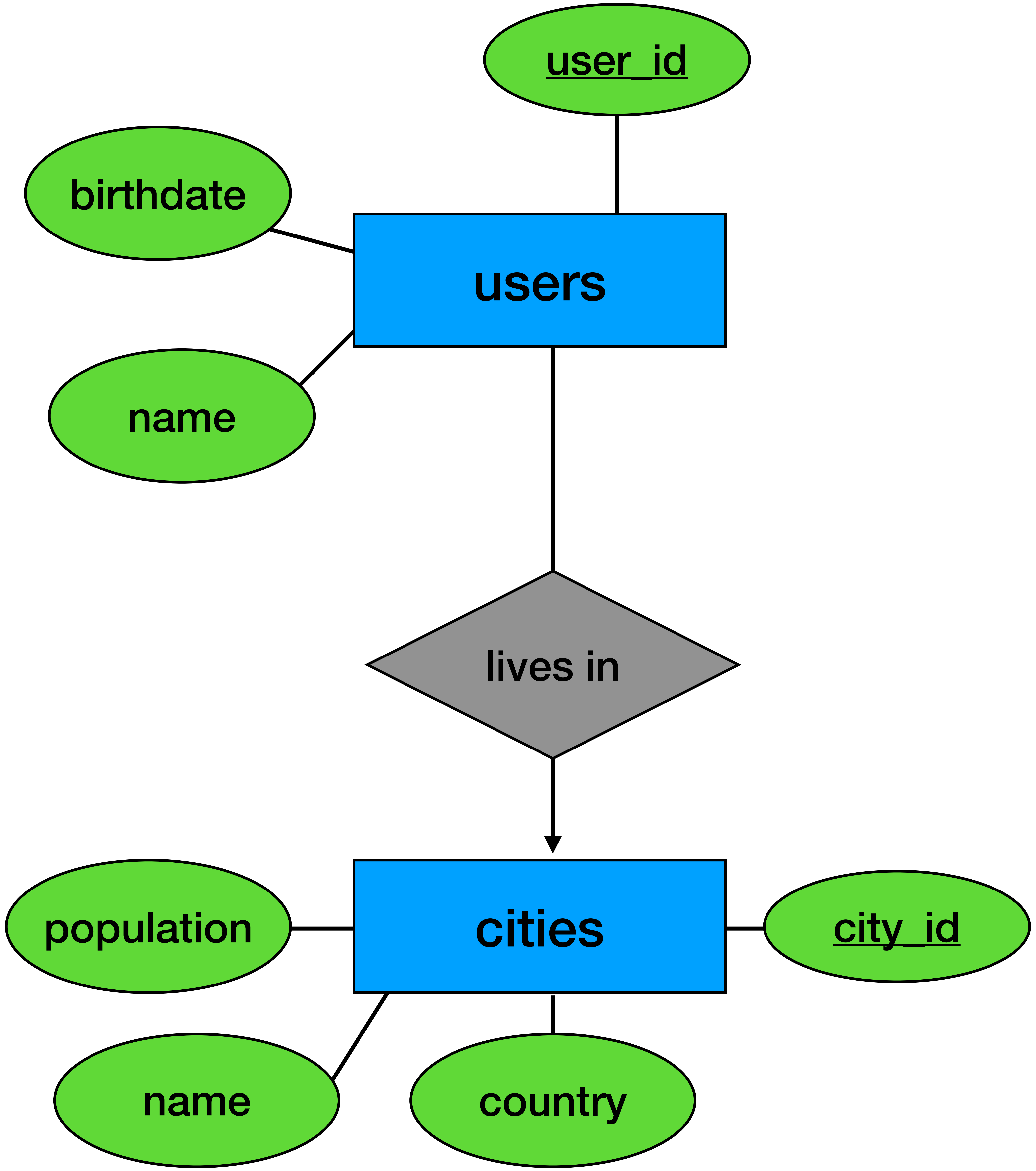
But **today** we are talking about **Relational DBs**


user_id
birthdate
users
city_id
country
name
city
population


user_id
birthdate
users
name


user_id
birthdate
users
name
population
cities
city_id
name
country


user_id
birthdate
users
name
lives in
population
cities
city_id
name
country


user_id

birthdate

users

name

lives in

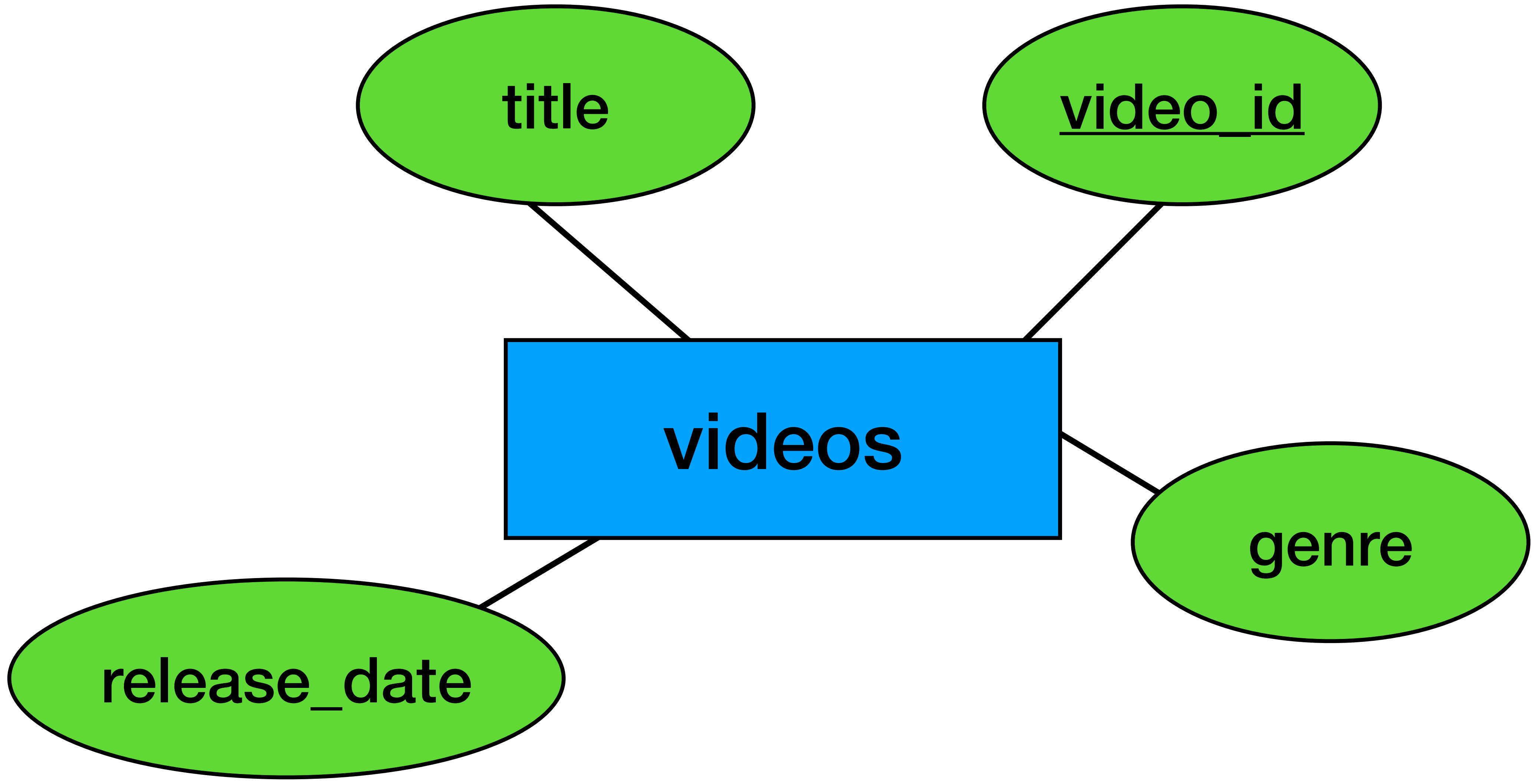
population

cities

city_id

name

country

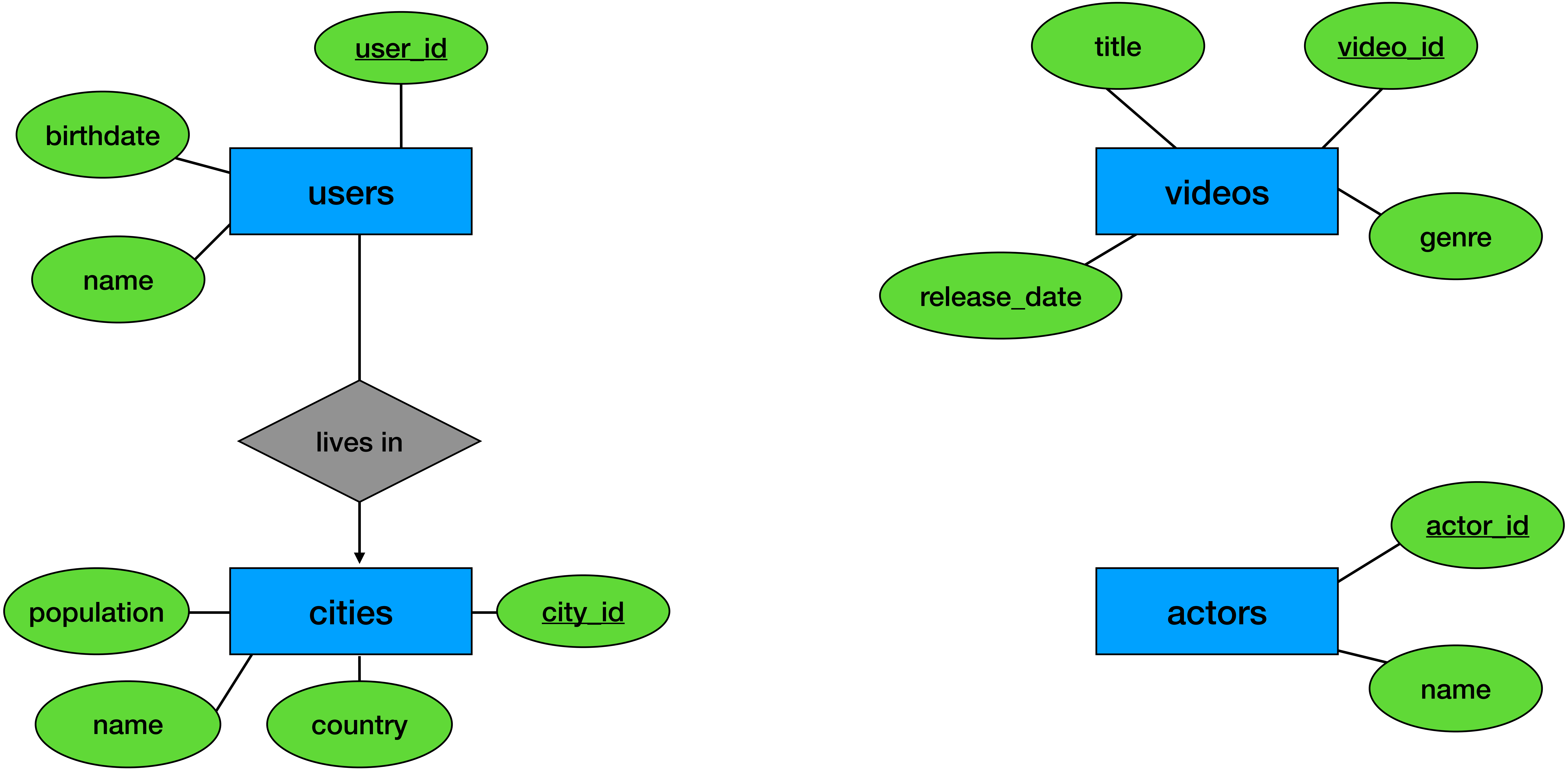

title

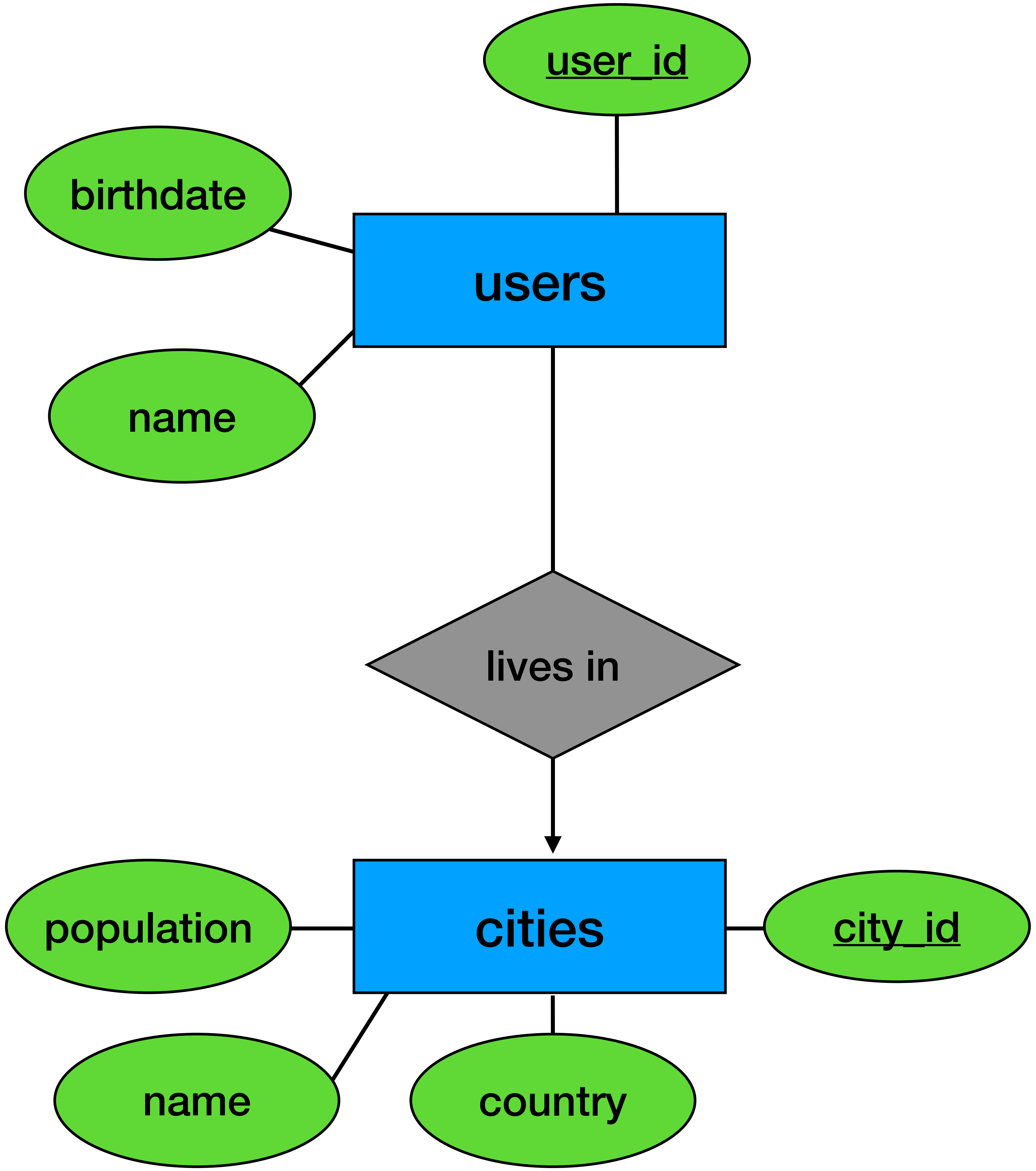
video_id

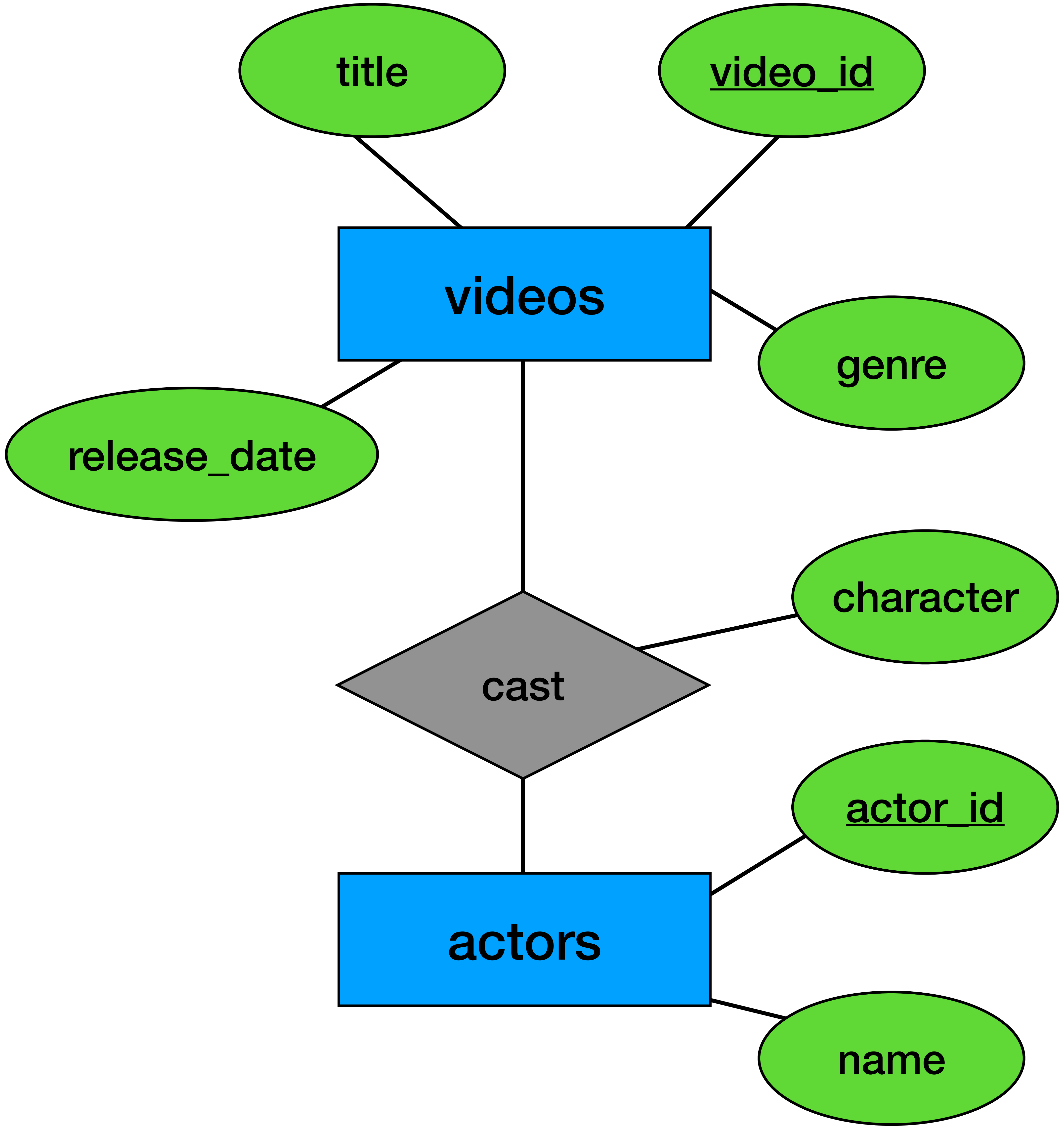
videos

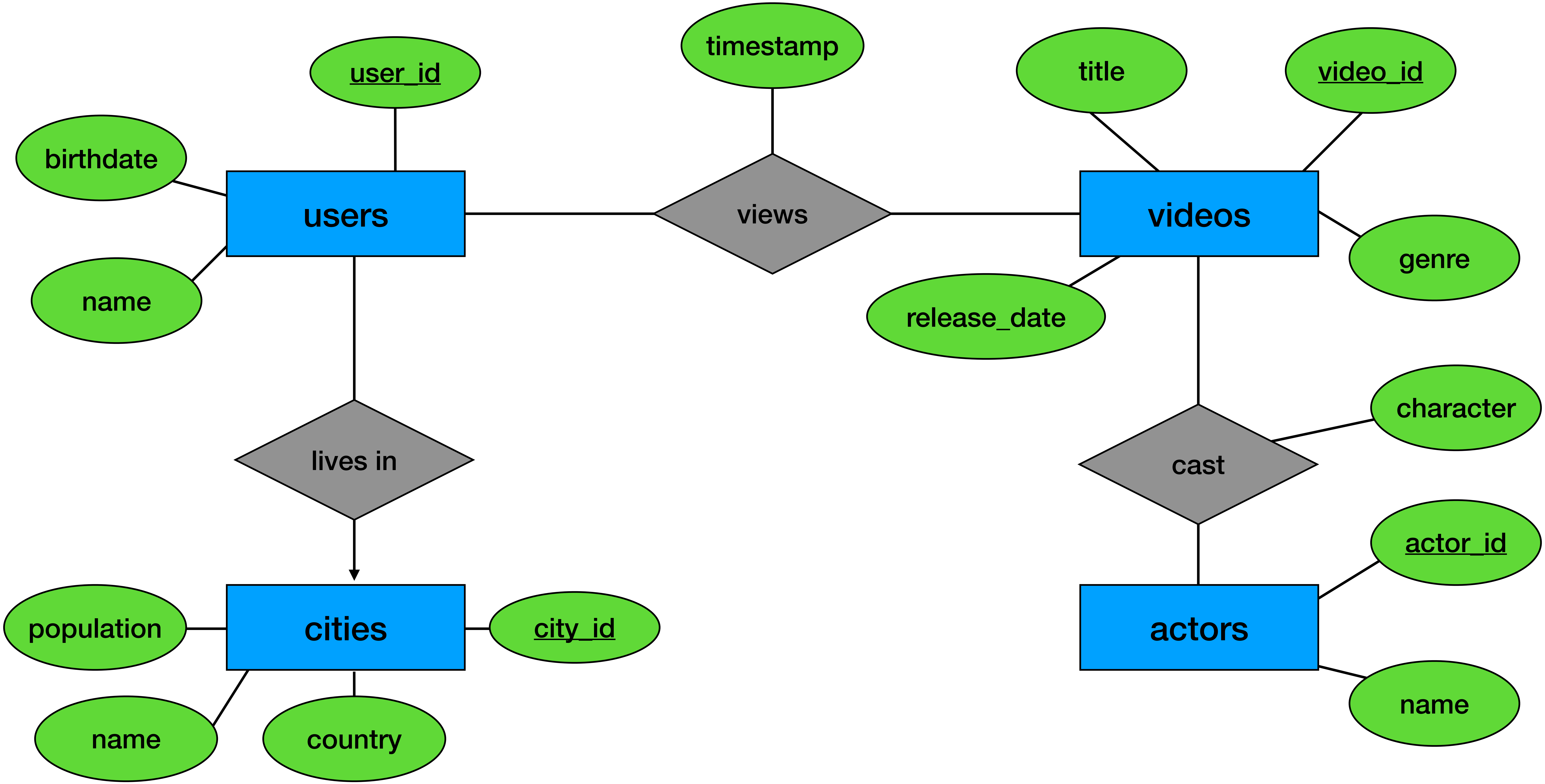
genre

release_date

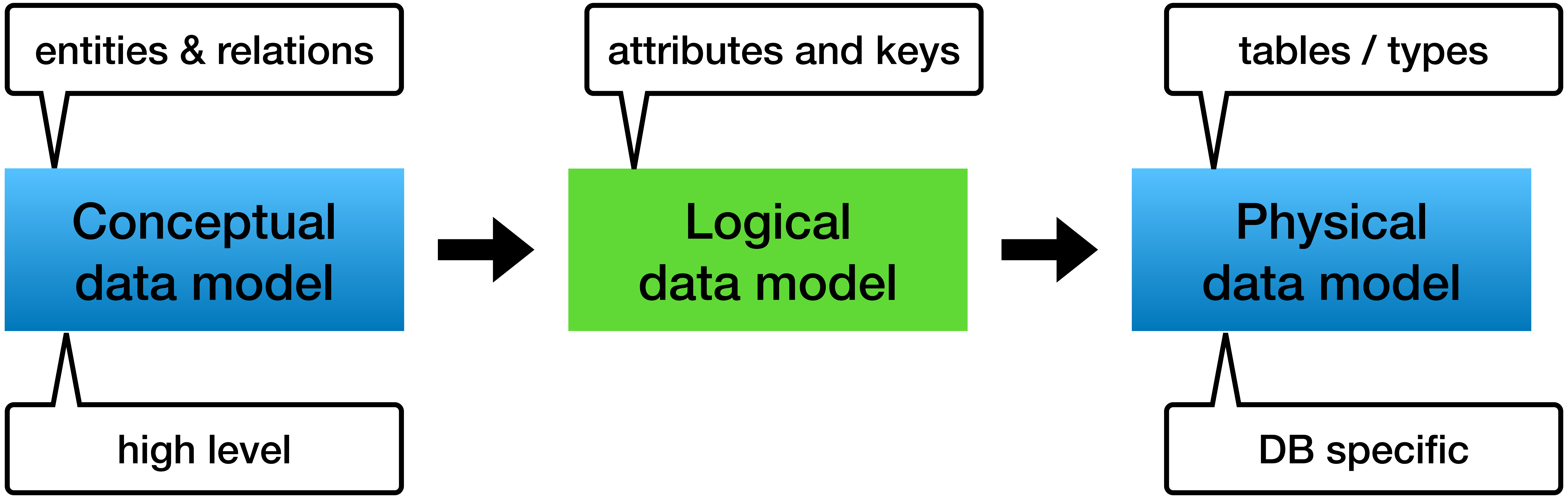

user_id
birthdate
users
name
lives in
population
cities
city_id
name
country
title
video_id
videos
genre
release_date
actor_id
actors
name


user_id
birthdate
users
name
lives in
population
cities
city_id
name
country


title
video_id
videos
genre
release_date
character
cast
actor_id
actors
name


user_id
birthdate
name
users
timestamp
views
title
video_id
videos
genre
release_date
lives in
character
cast
actor_id
population
cities
city_id
actors
name
country
name

Relational Modeling - 10,000 foot view

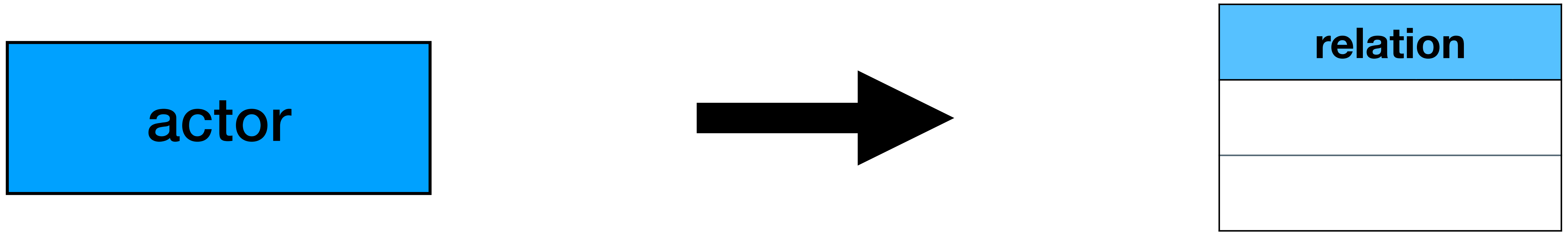


Logical data model

- From concept the “schema”
- Keys, foreign keys
- Data types are not yet defined

ER to Relational schema

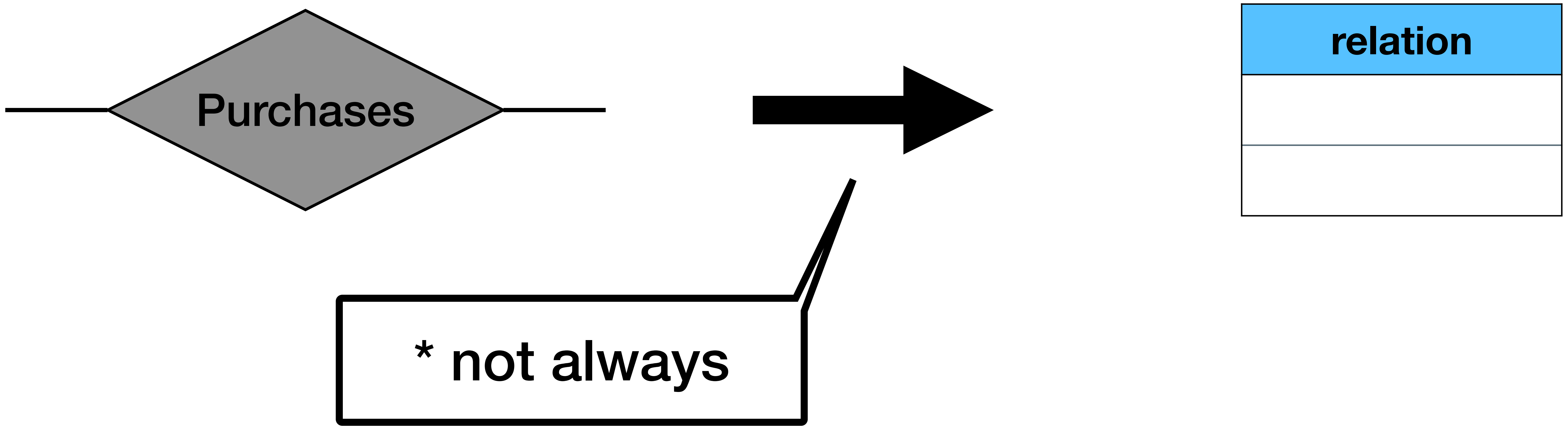
- Entities



actor

relation

- Relations

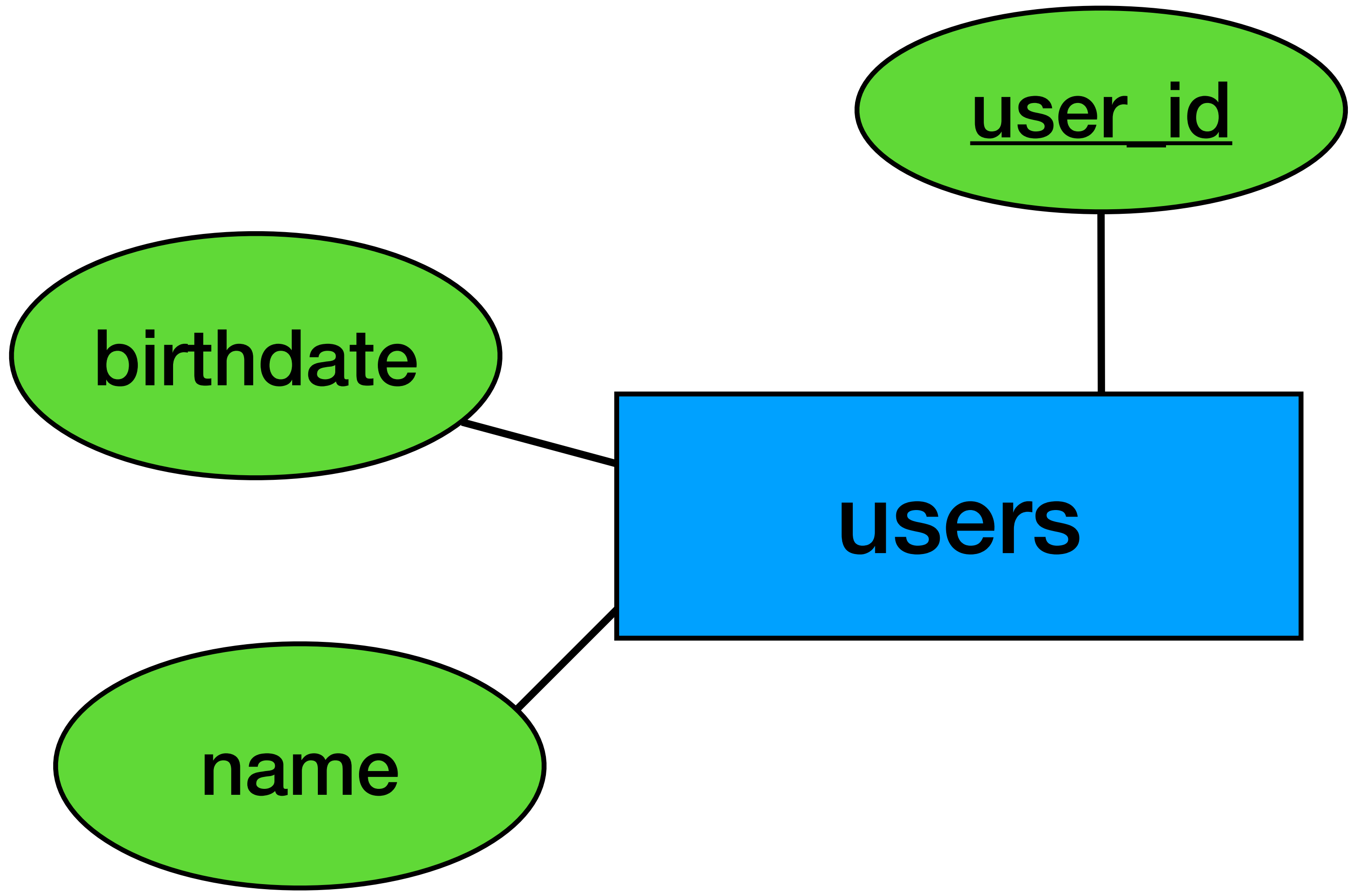


Purchases

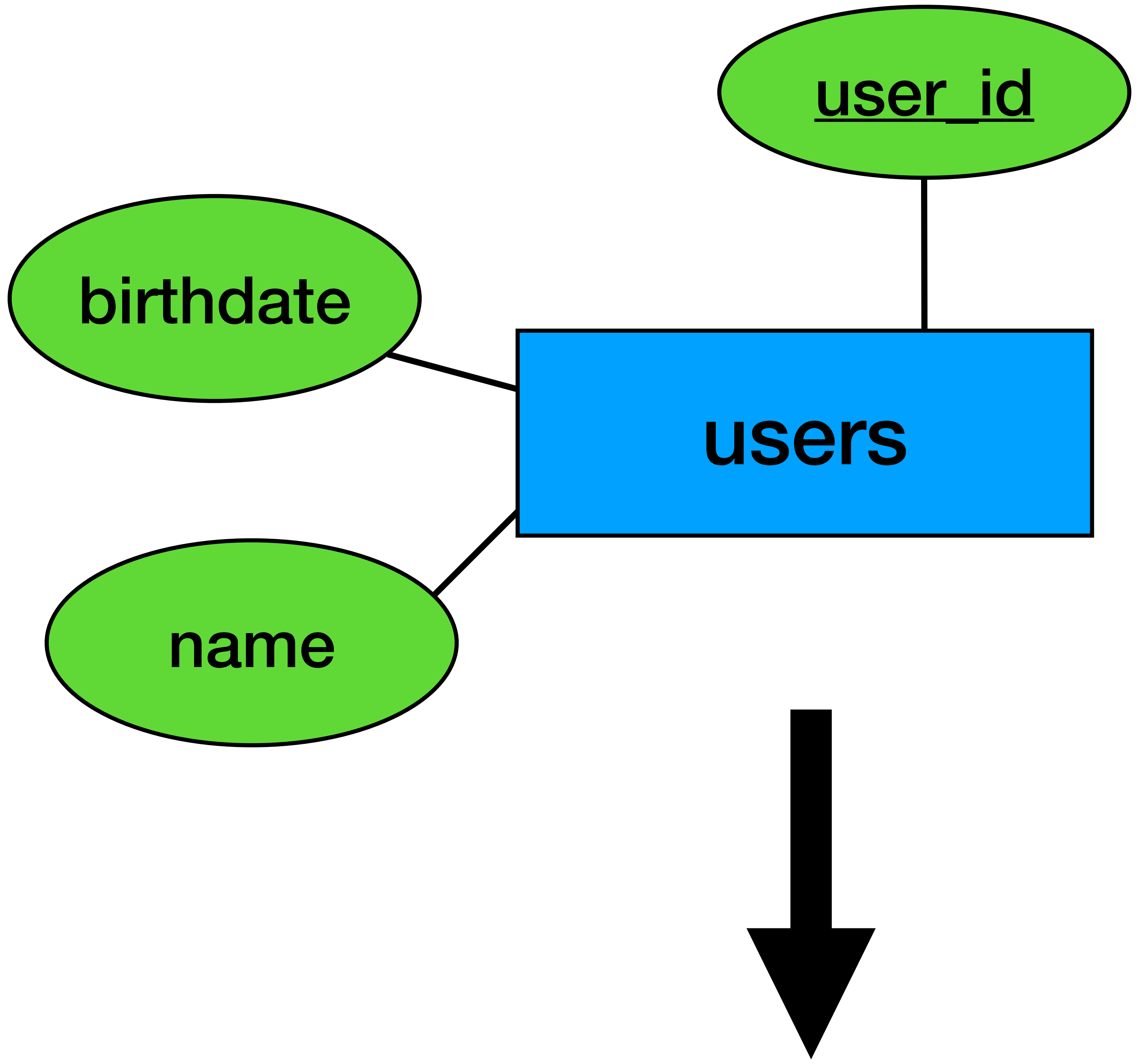
relation

* not always

Entity to Relation

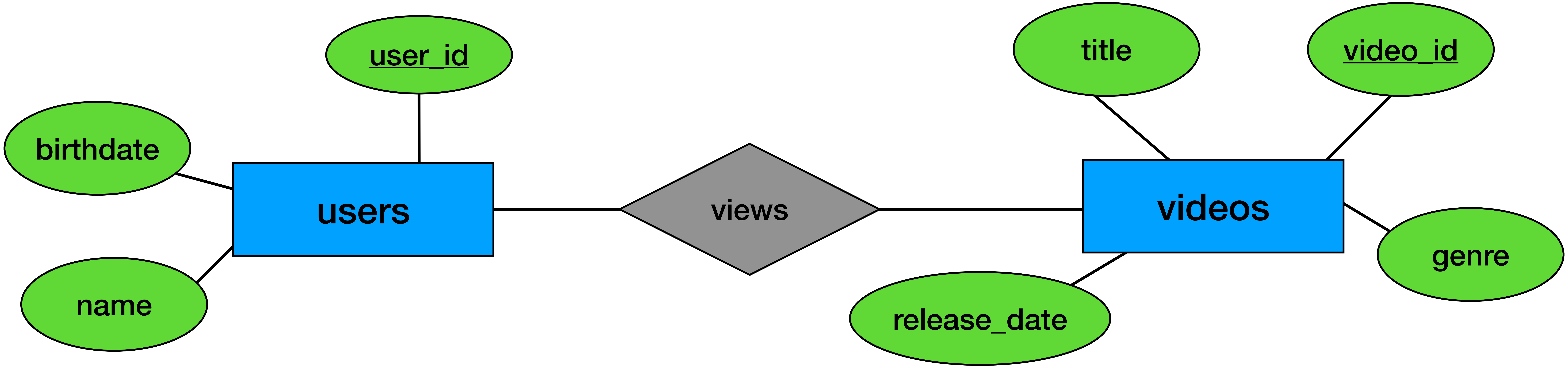


Entity to Relation

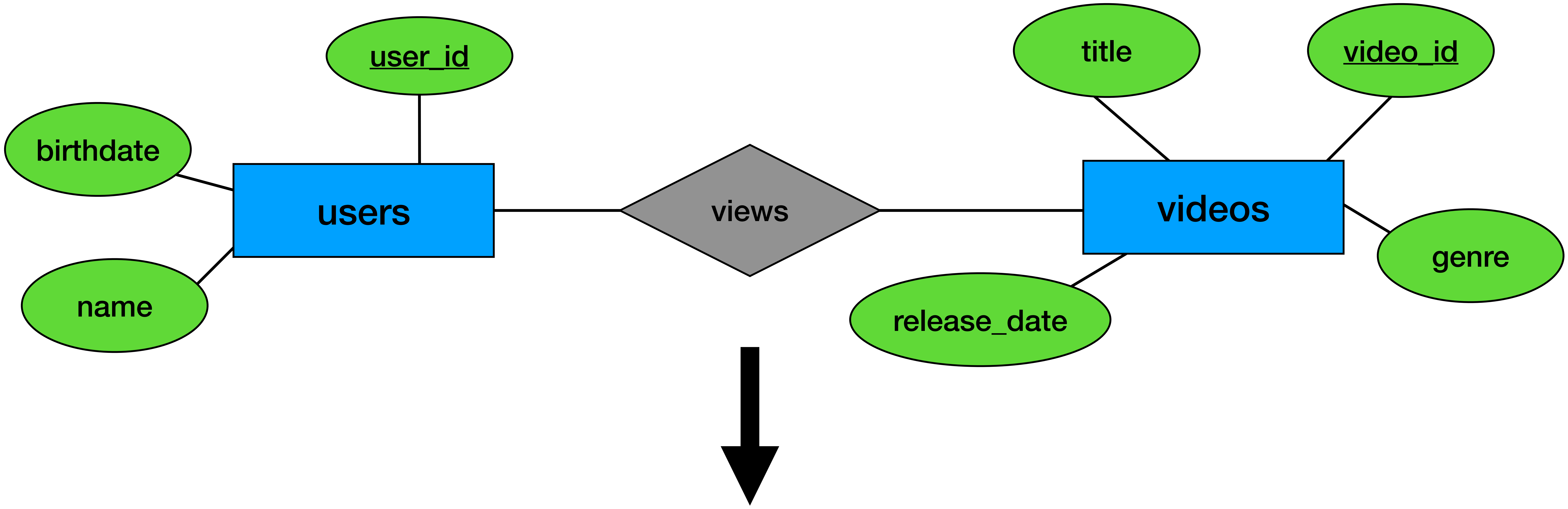


users	
user_id	K
name	
birthdate	

Relation to Relation (many-to-many)



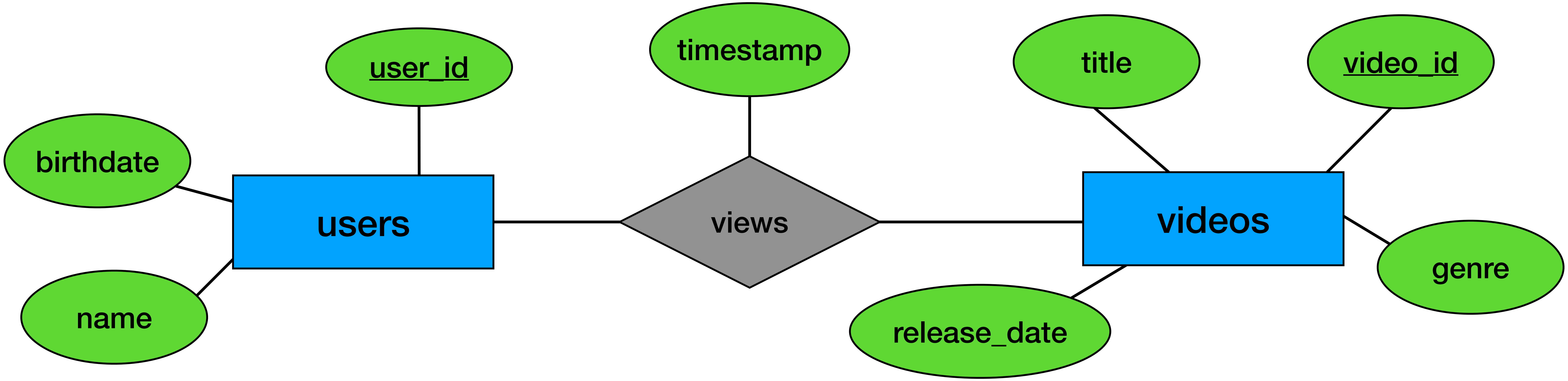
Relation to Relation (many-to-many)



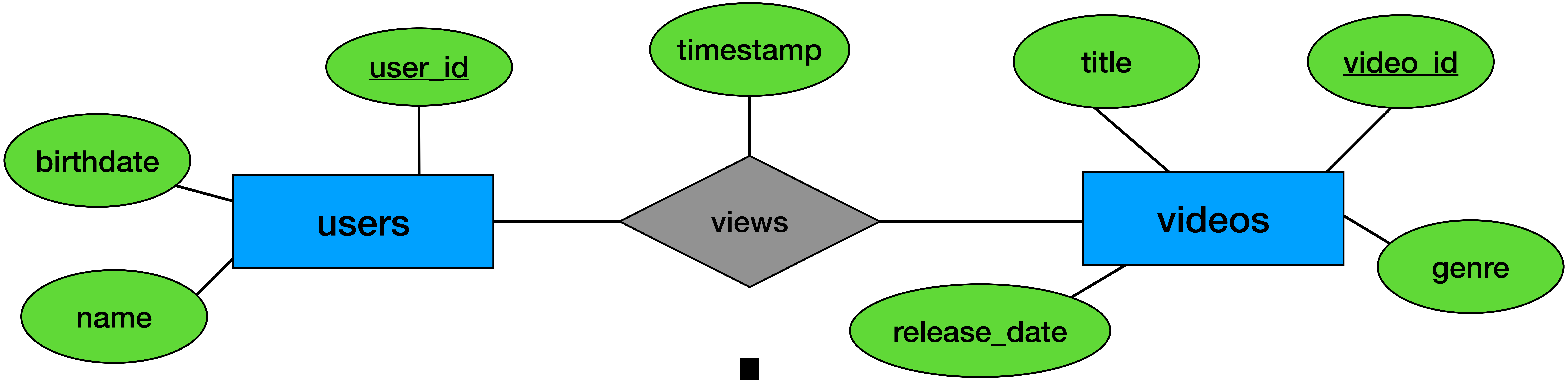
views	
user_id	K
video_id	K

Keys are derived from entities

Relation to Relation (+attributes)



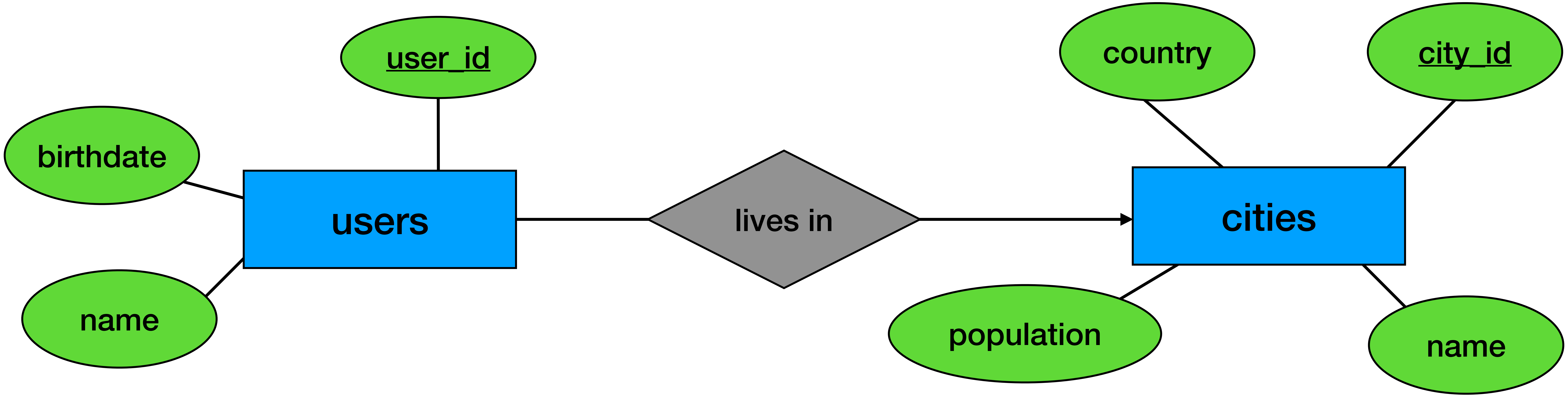
Relation to Relation (+attributes)



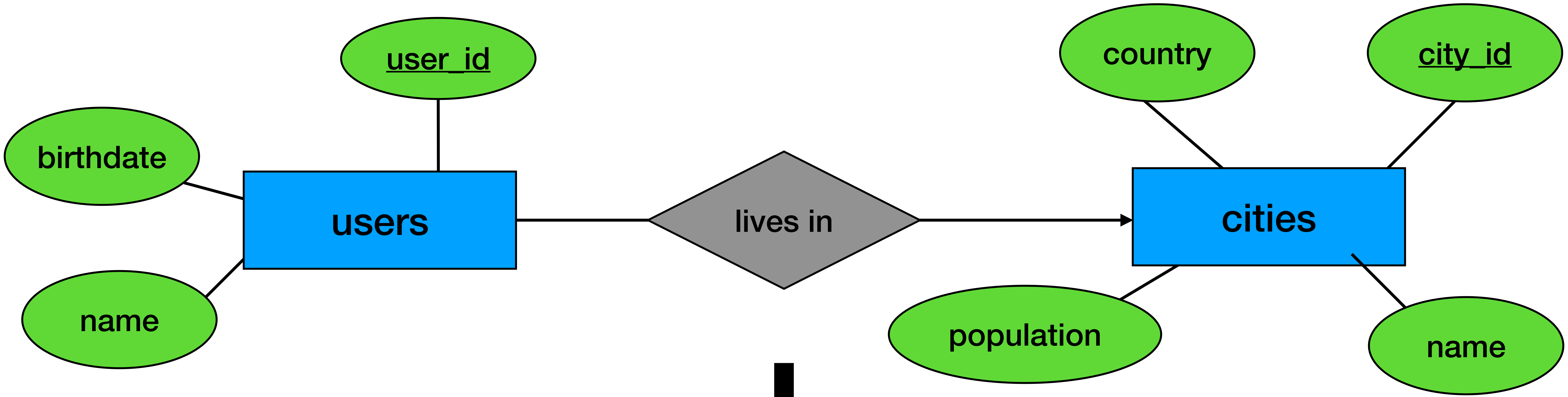
views	
user_id	K
video_id	K
timestamp	

Additional attributes

Relation to Relation (many-to-one)



Relation to Relation (many-to-one)

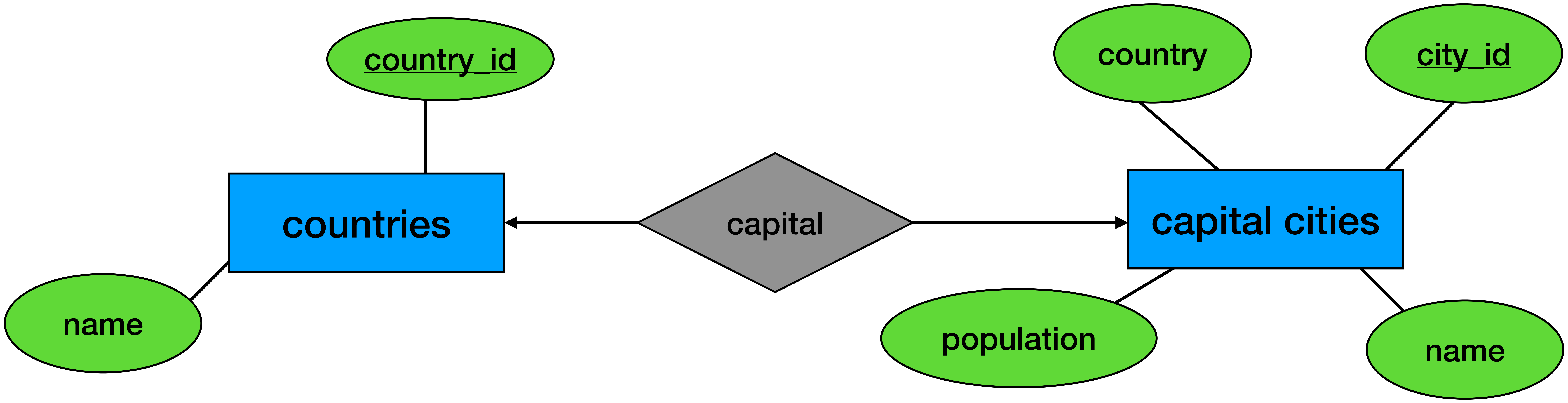


users	
user_id	K
name	
birthdate	
city_id	FK

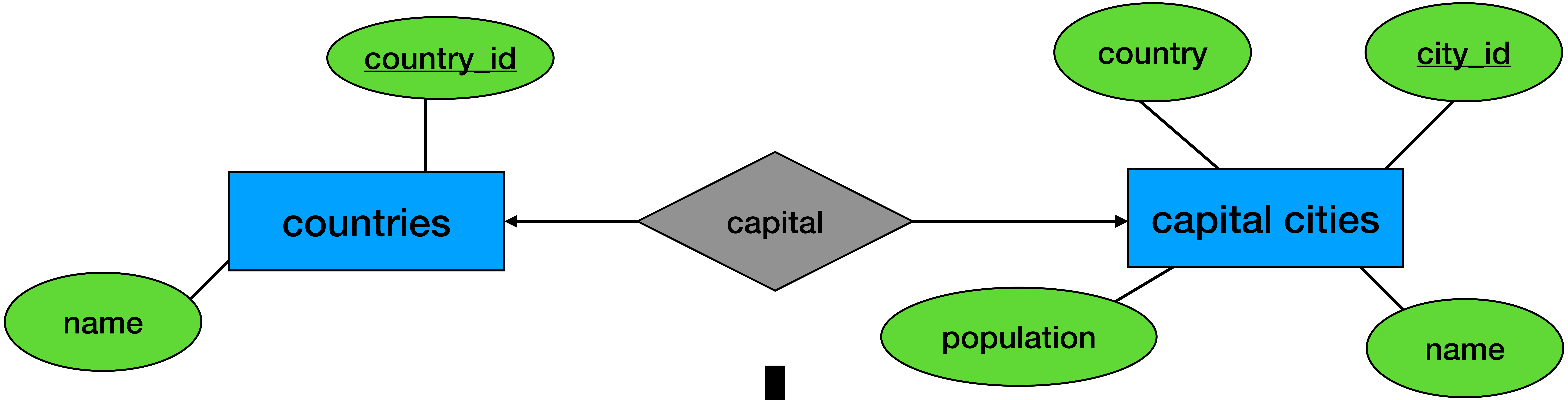
No additional table is required. We add to users the key(s) of cities

Sometimes FKs are omitted and will be represented by arrows (see next slides)

Relation to Relation (one-to-one)



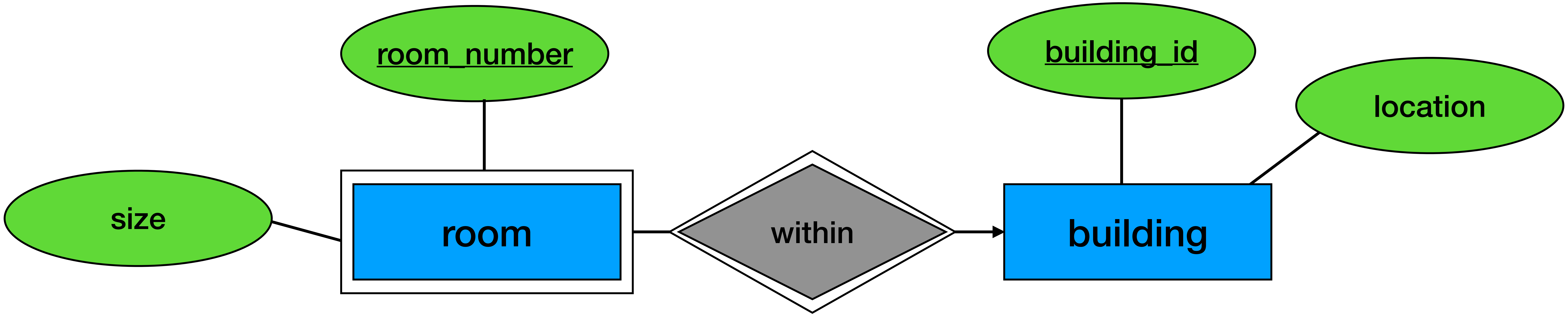
Relation to Relation (one-to-one)



countries	
country_id	K
name	
city_id	FK, U

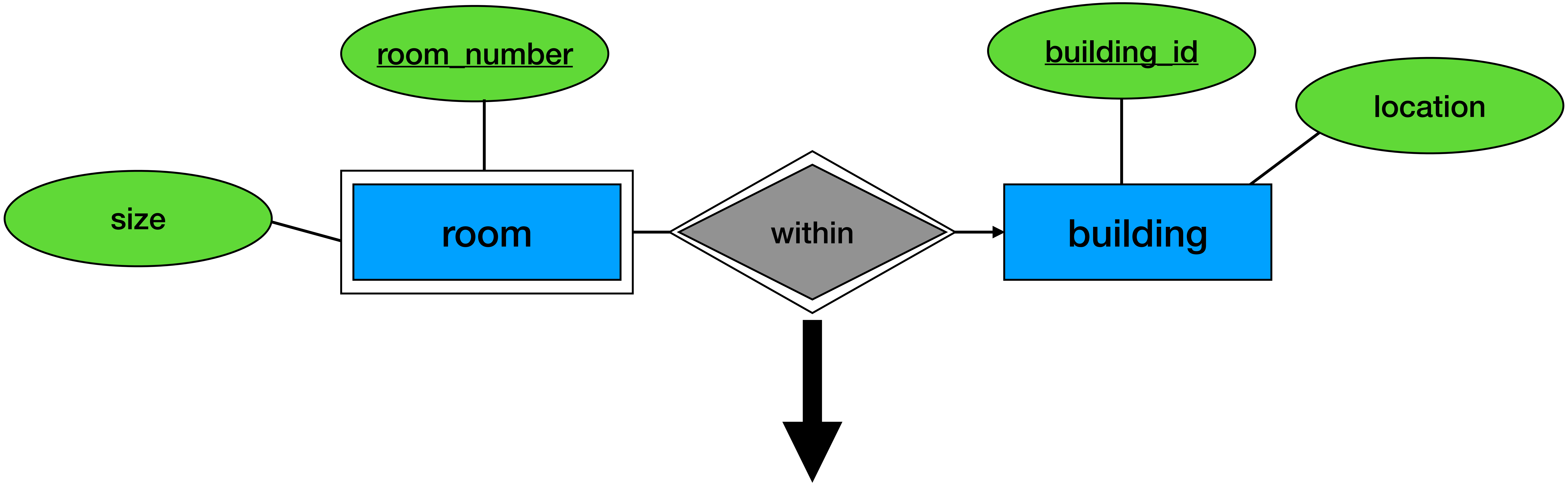
FK + Unique index

Weak Entity



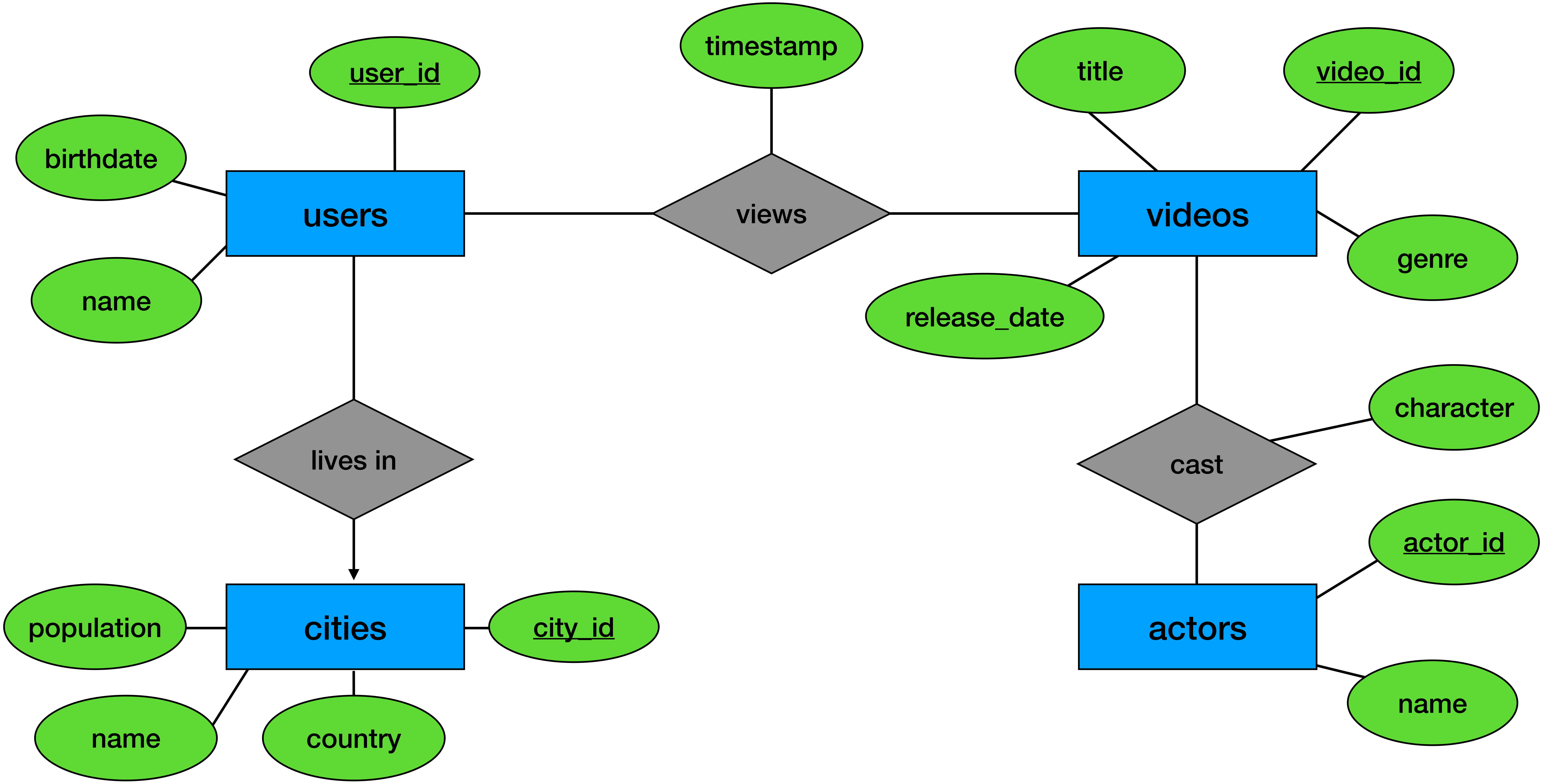
room_number
building_id
location
size
room
within
building

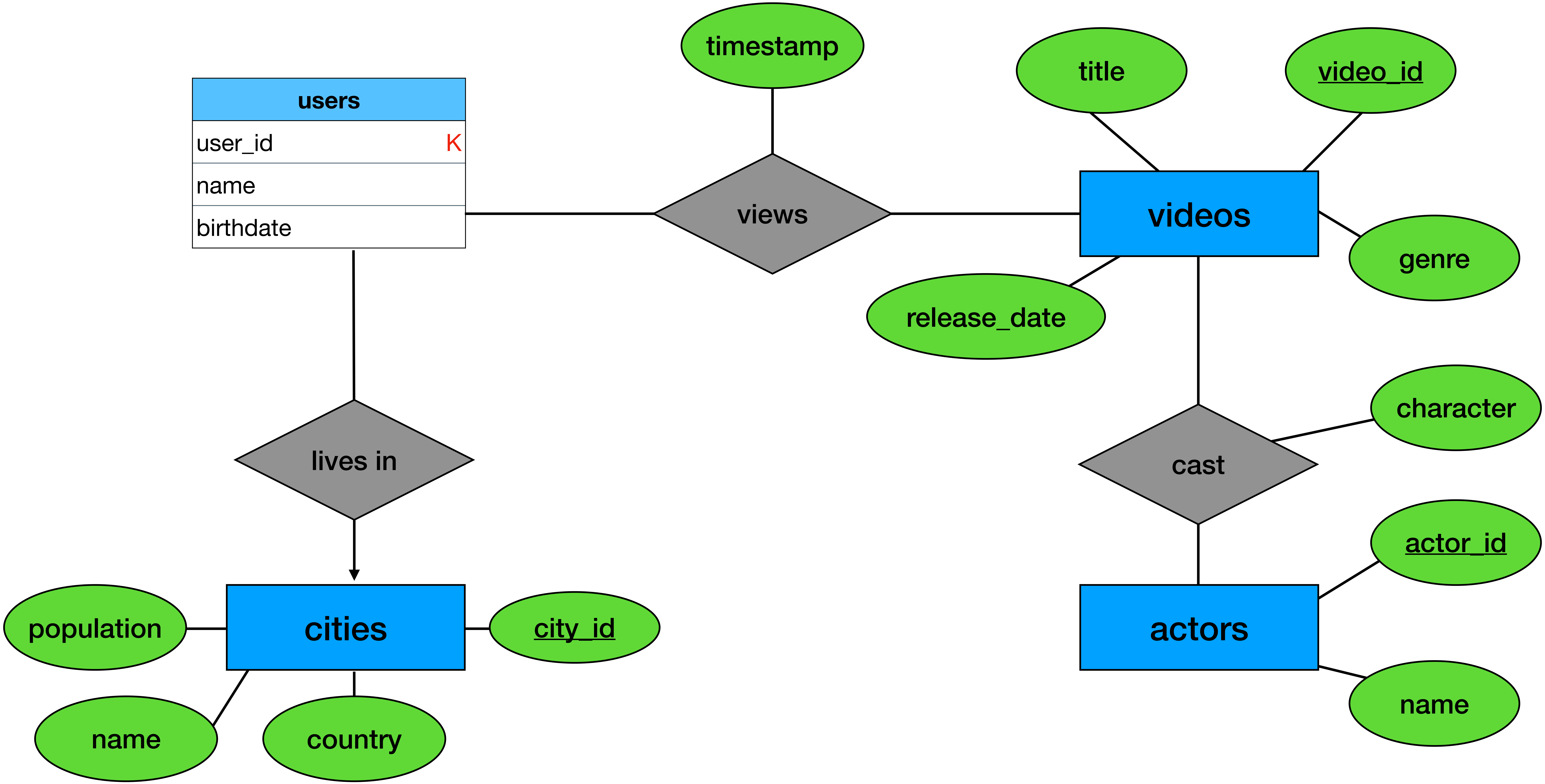
Weak Entity

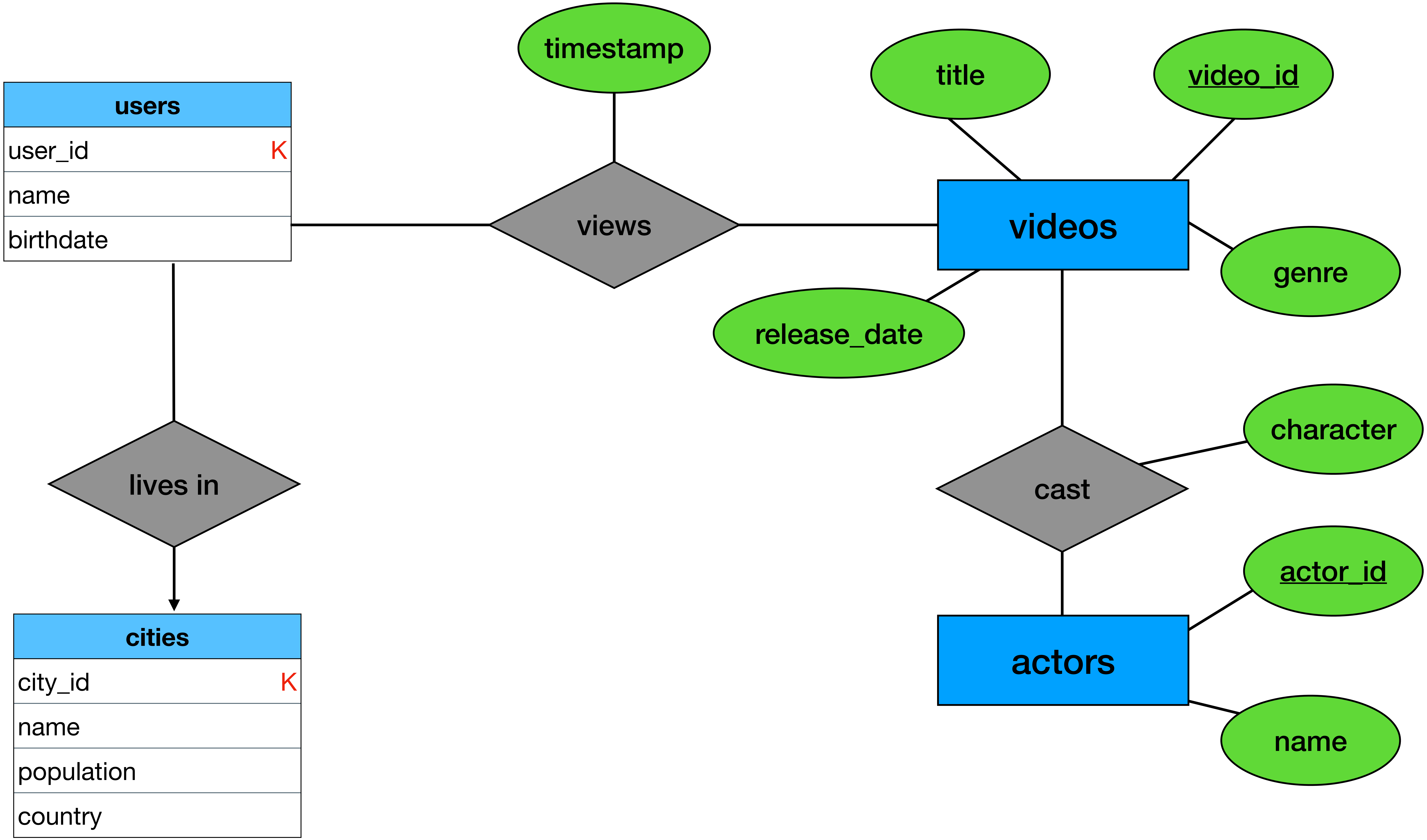


room	
building_id	K
room_number	K
size	

Example


user_id
timestamp
title
video_id
birthdate
users
views
videos
genre
name
release_date
character
lives in
cast
actor_id
population
cities
city_id
actors
name
country
name


timestamp
title
video_id
users
user_id
K
name
birthdate
views
videos
genre
release_date
character
lives in
cast
actor_id
population
cities
city_id
actors
name
country
name



users	
user_id	K
name	
birthdate	

cities	
city_id	K
name	
population	
country	

timestamp

views

lives in

title

video_id

videos

genre

release_date

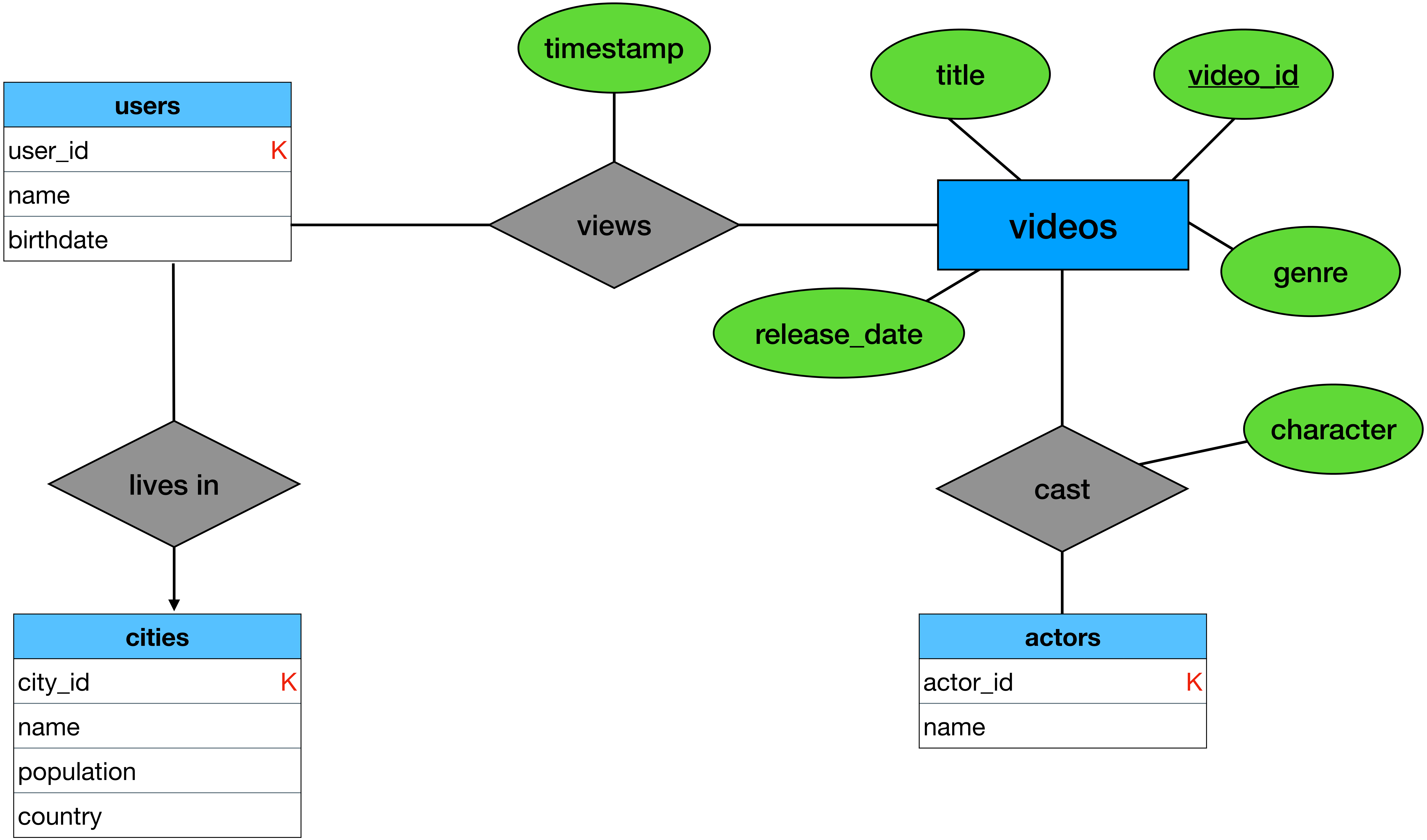
character

cast

actor_id

actors

name



users
- user_id K
- name
- birthdate

views
timestamp

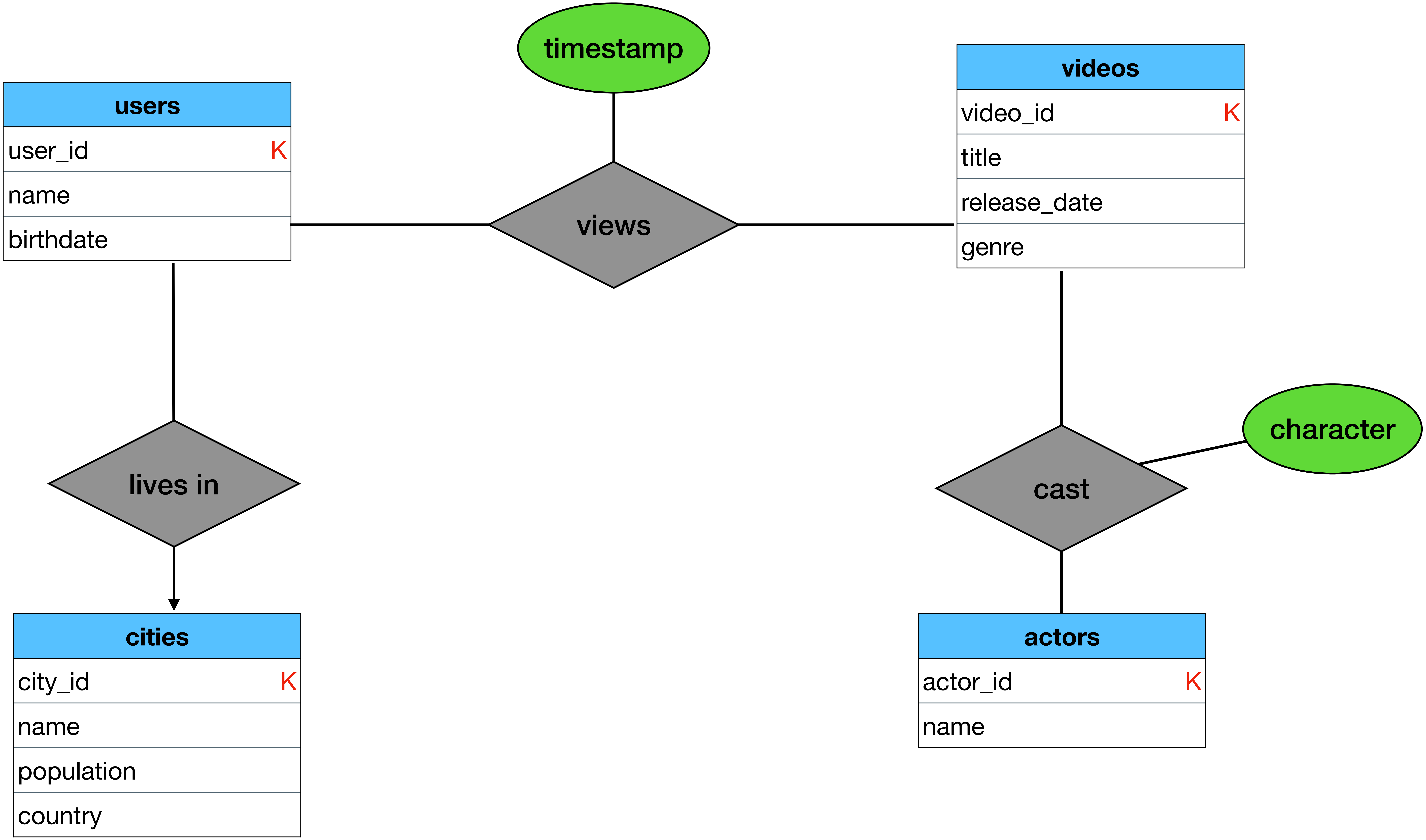
videos
title
video_id
genre
release_date

lives in

cities
- city_id K
- name
- population
- country

cast
character

actors
- actor_id K
- name



users

- user_id
- name
- birthdate

videos

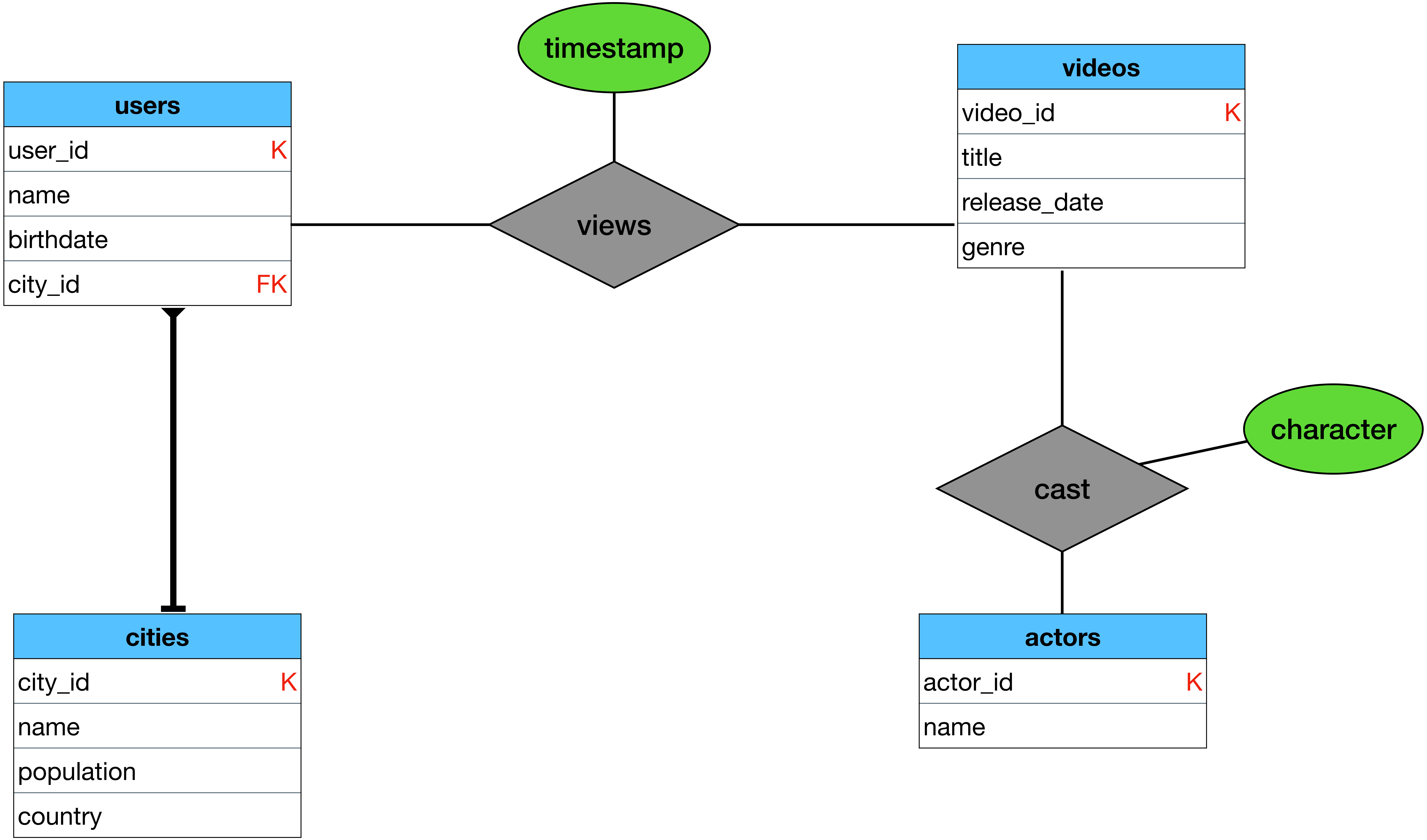
- video_id
- title
- release_date
- genre

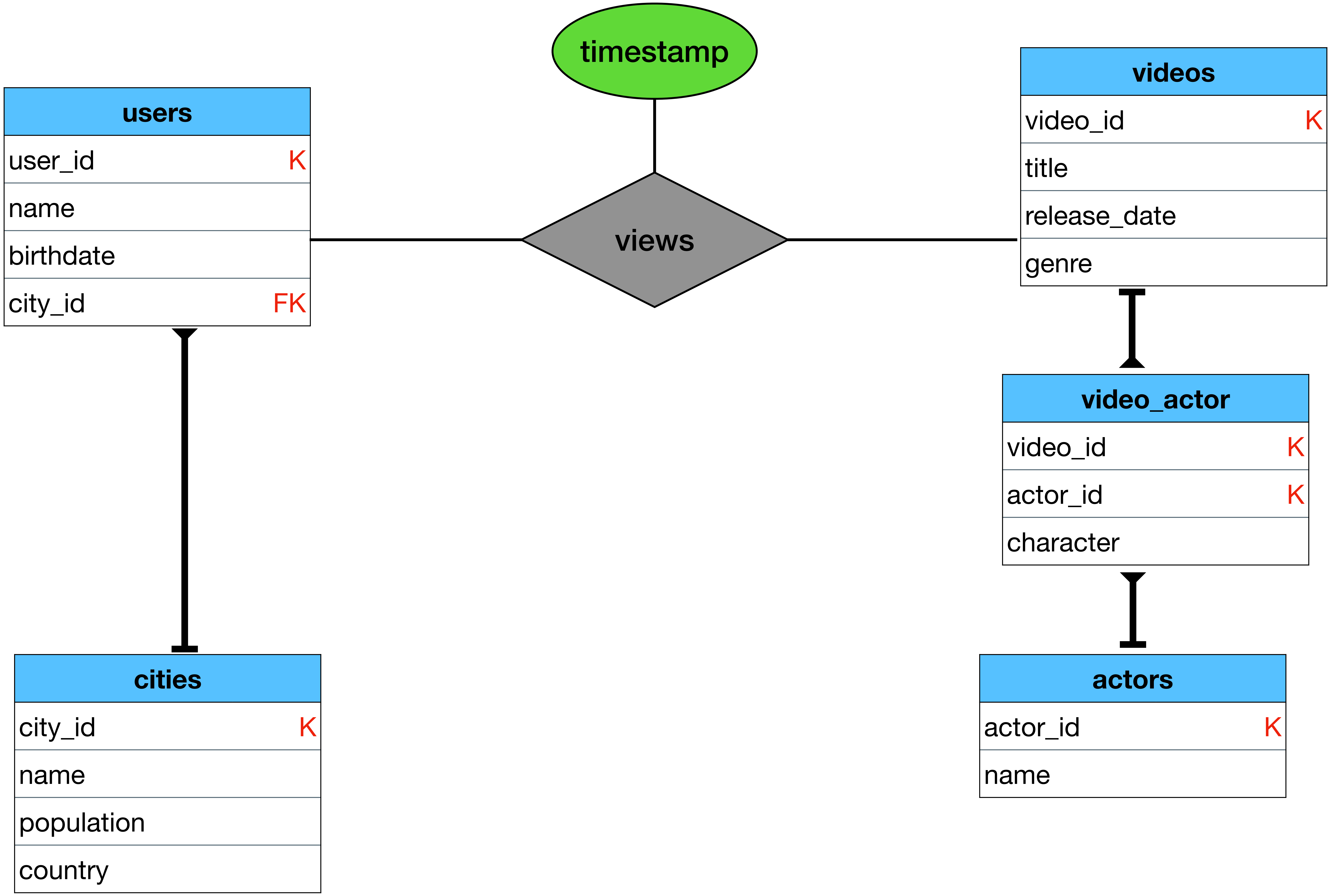
cities

- city_id
- name
- population
- country

actors

- actor_id
- name


timestamp
users
user_id K
name
birthdate
city_id FK
views
videos
video_id K
title
release_date
genre
character
cast
cities
city_id K
name
population
country
actors
actor_id K
name



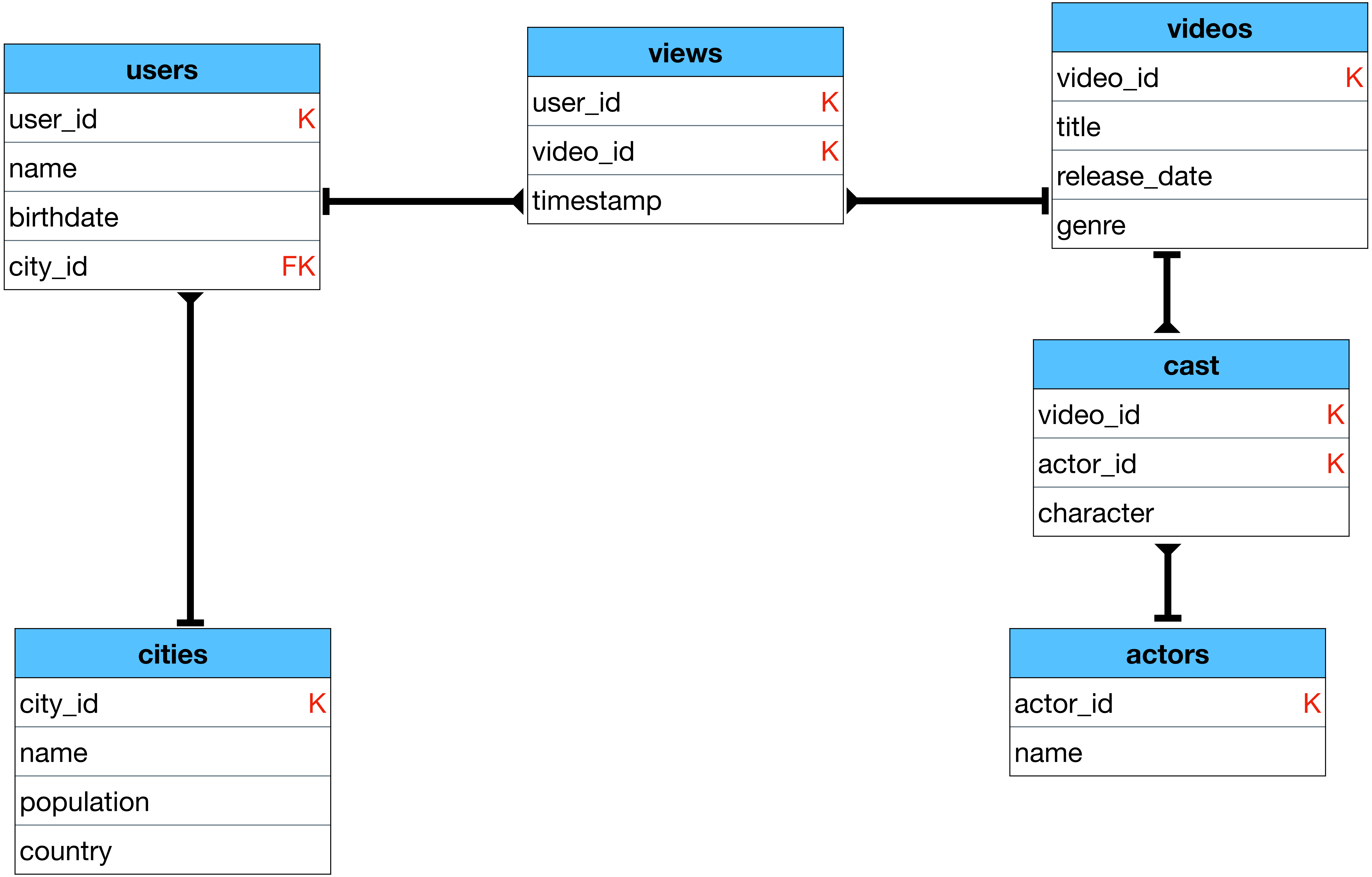
users	
user_id	K
name	
birthdate	
city_id	FK

cities	
city_id	K
name	
population	
country	

videos	
video_id	K
title	
release_date	
genre	

video_actor	
video_id	K
actor_id	K
character	

actors	
actor_id	K
name	



users	
user_id	K
name	
birthdate	
city_id	FK

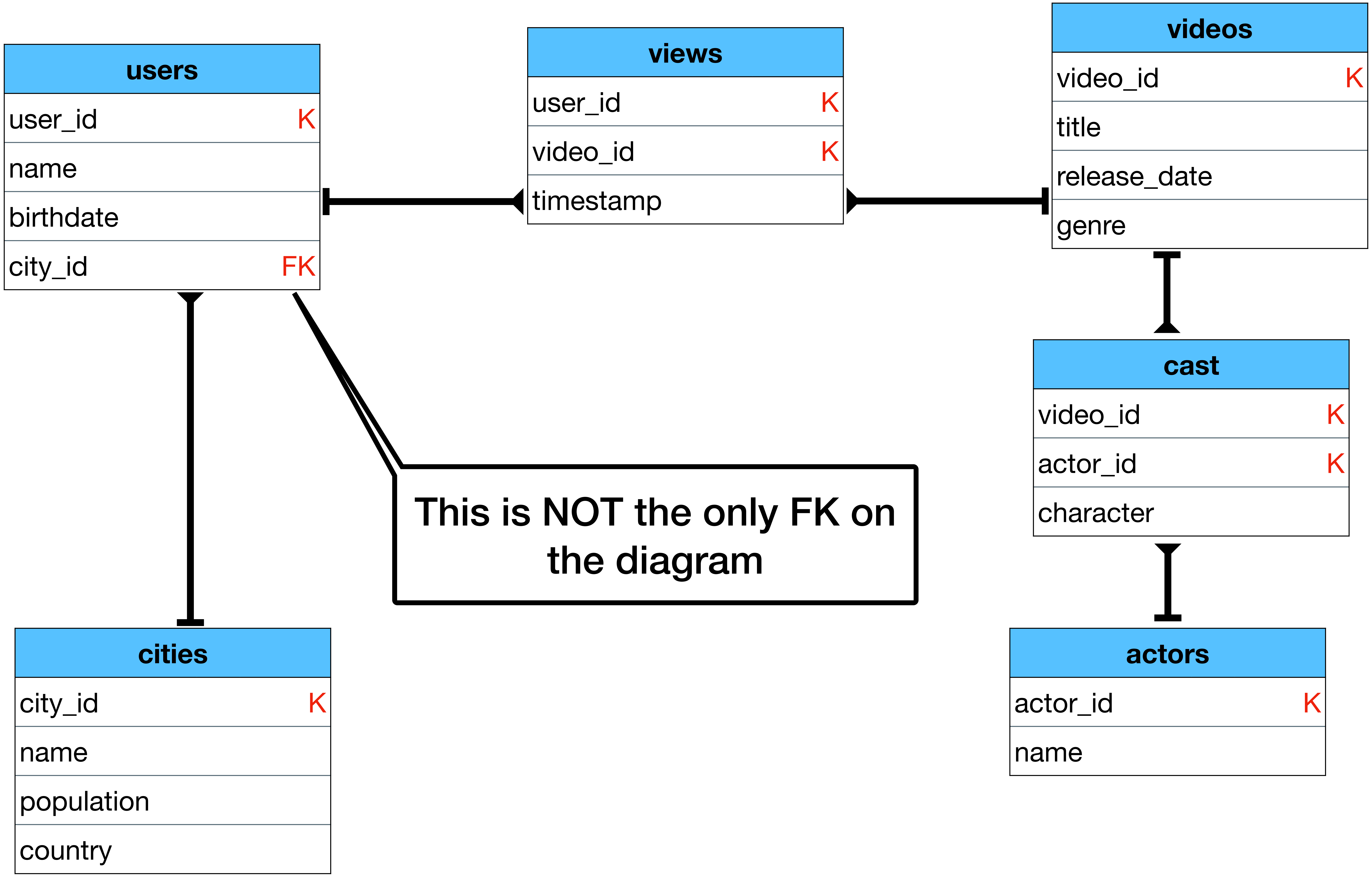
views	
user_id	K
video_id	K
timestamp	

videos	
video_id	K
title	
release_date	
genre	

cast	
video_id	K
actor_id	K
character	

cities	
city_id	K
name	
population	
country	

actors	
actor_id	K
name	



users	
user_id	K
name	
birthdate	
city_id	FK

views	
user_id	K
video_id	K
timestamp	

videos	
video_id	K
title	
release_date	
genre	

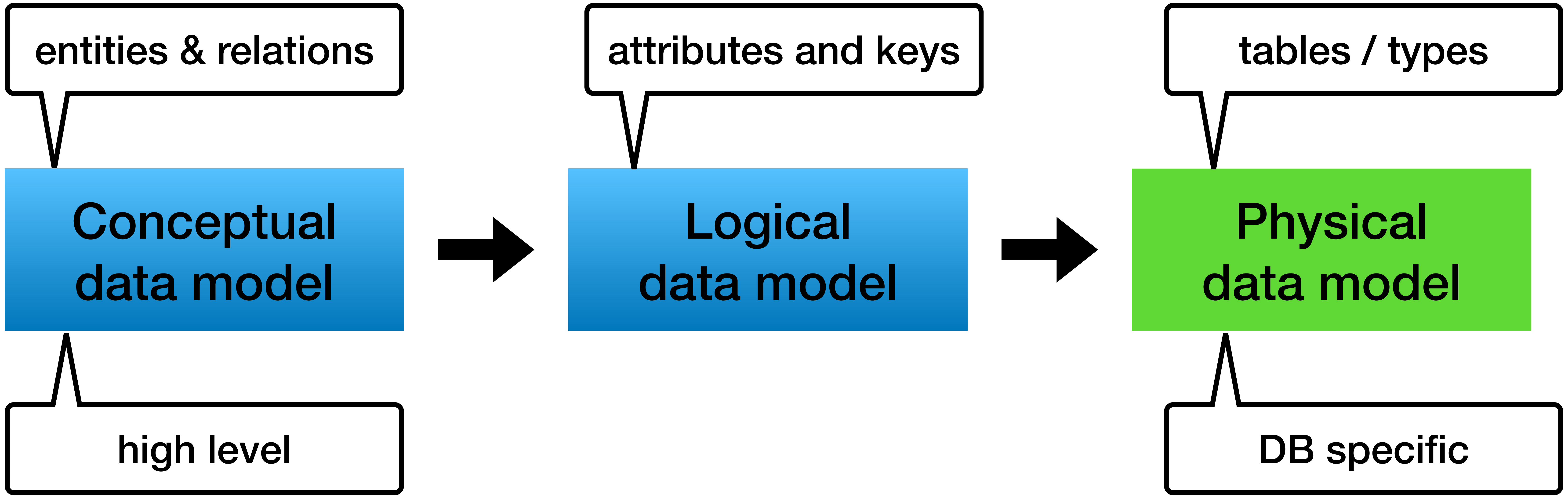
cast	
video_id	K
actor_id	K
character	

cities	
city_id	K
name	
population	
country	

actors	
actor_id	K
name	

This is NOT the only FK on the diagram

Relational Modeling - 10,000 foot view



entities & relations

Conceptual data model

high level

attributes and keys

Logical data model

tables / types

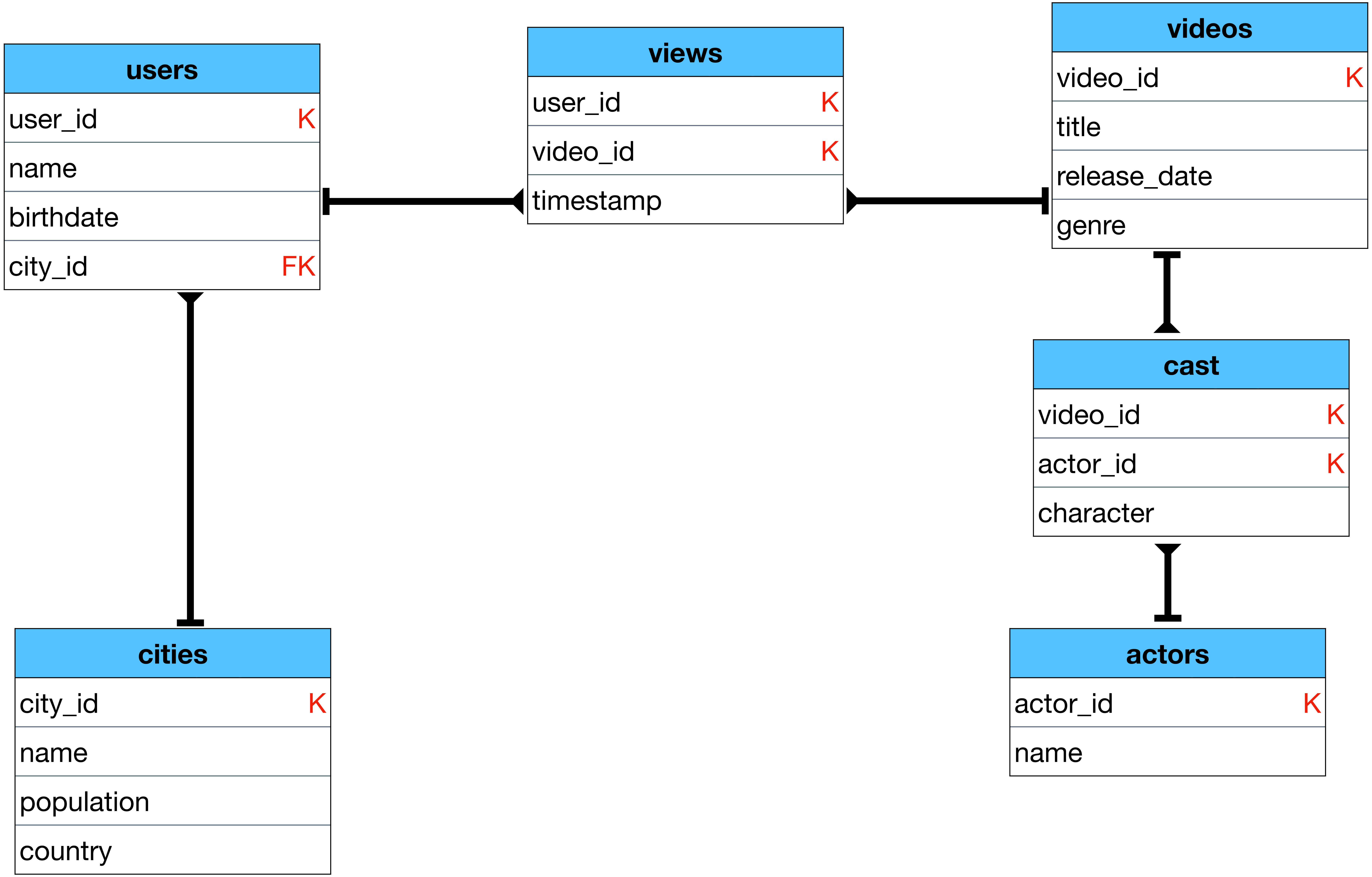
Physical data model

DB specific

Physical data model

- Finalize the schema
- Add types
- Generate create table statements

Example



users	
user_id	K
name	
birthdate	
city_id	FK

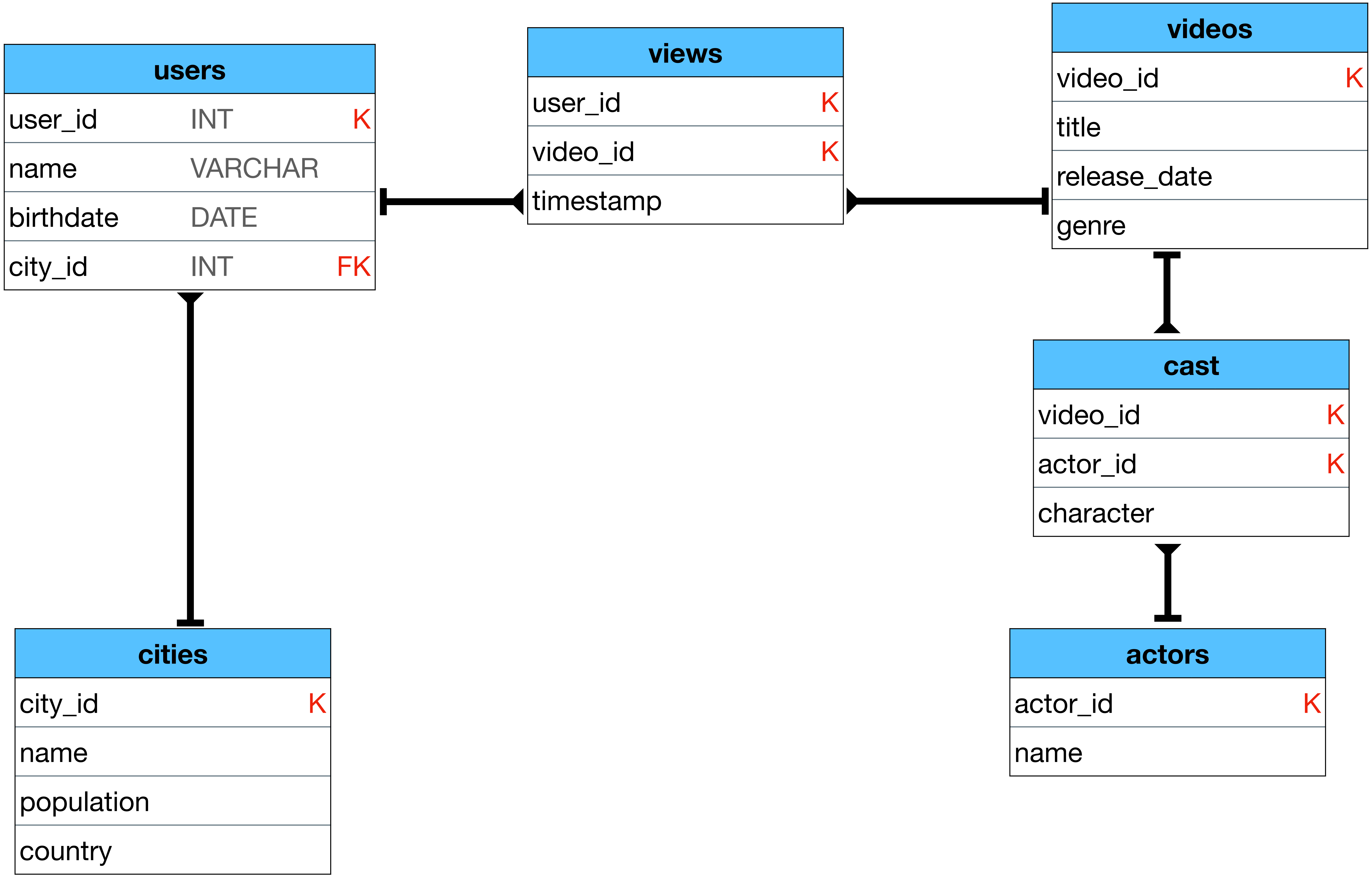
views	
user_id	K
video_id	K
timestamp	

videos	
video_id	K
title	
release_date	
genre	

cast	
video_id	K
actor_id	K
character	

cities	
city_id	K
name	
population	
country	

actors	
actor_id	K
name	



users		
user_id	INT	K
name	VARCHAR	
birthdate	DATE	
city_id	INT	FK

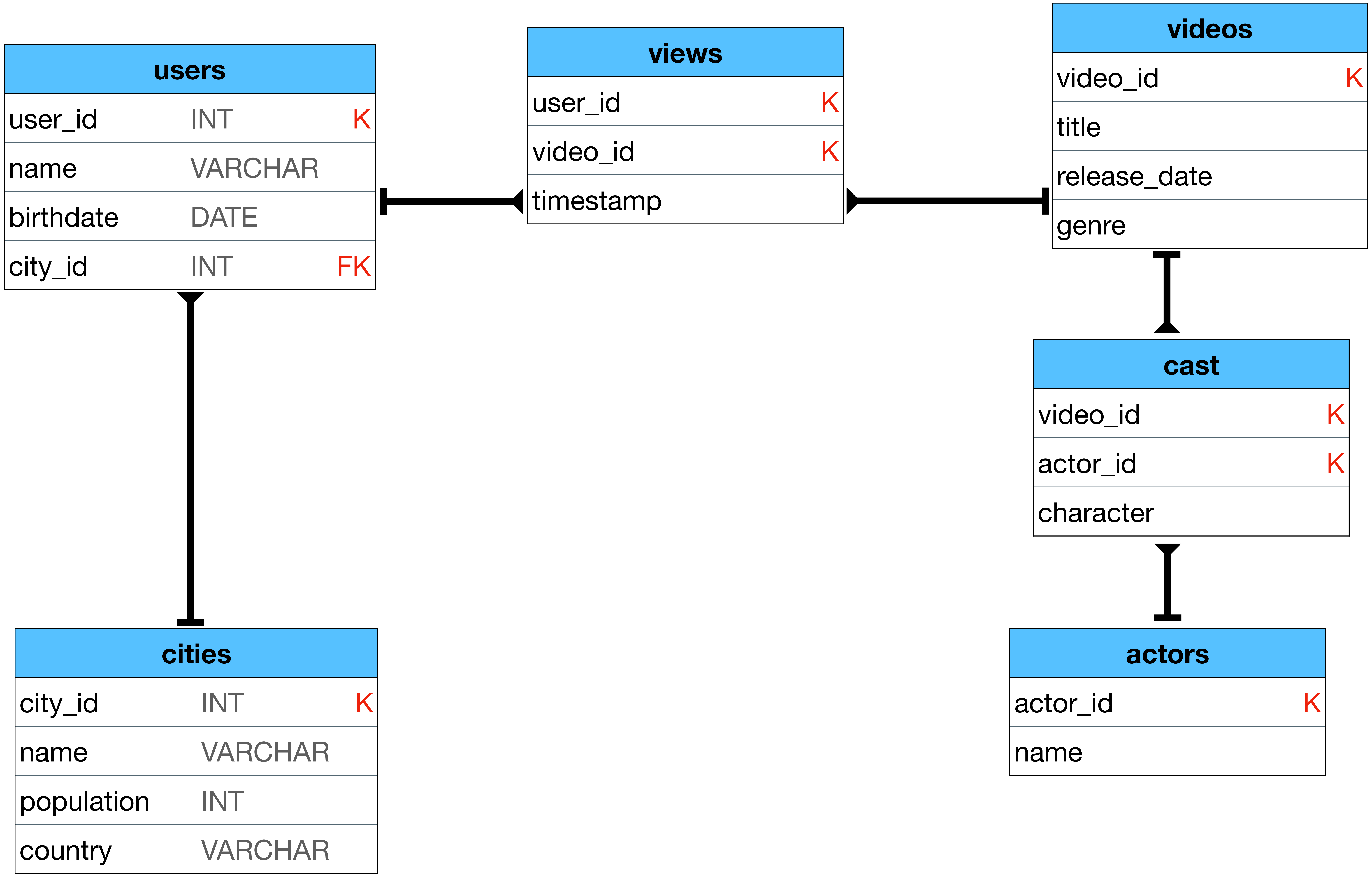
views	
user_id	K
video_id	K
timestamp	

videos	
video_id	K
title	
release_date	
genre	

cast	
video_id	K
actor_id	K
character	

cities	
city_id	K
name	
population	
country	

actors	
actor_id	K
name	



users		
user_id	INT	K
name	VARCHAR	
birthdate	DATE	
city_id	INT	FK

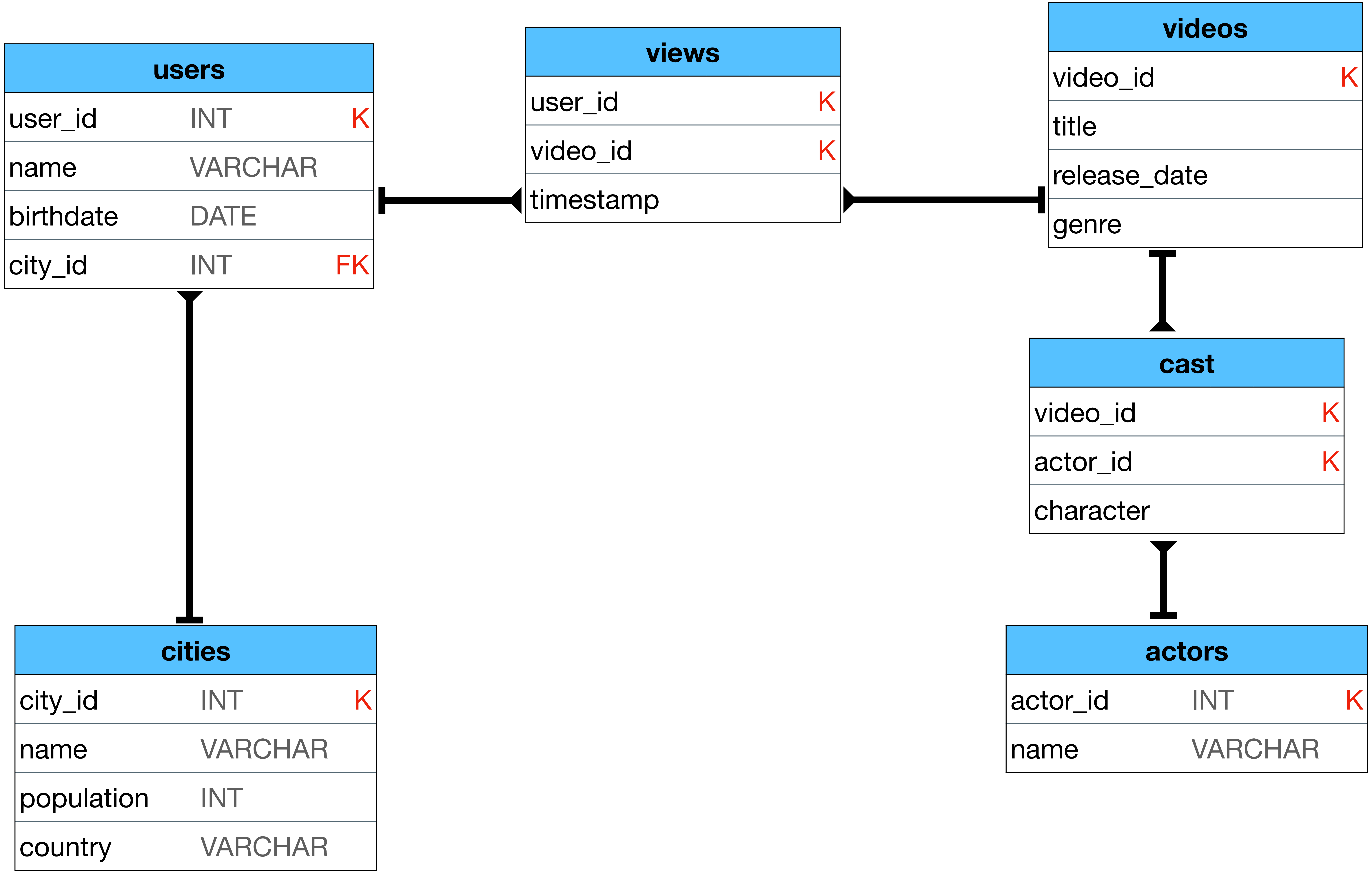
views	
user_id	K
video_id	K
timestamp	

videos	
video_id	K
title	
release_date	
genre	

cast	
video_id	K
actor_id	K
character	

cities		
city_id	INT	K
name	VARCHAR	
population	INT	
country	VARCHAR	

actors	
actor_id	K
name	



users		
user_id	INT	K
name	VARCHAR	
birthdate	DATE	
city_id	INT	FK

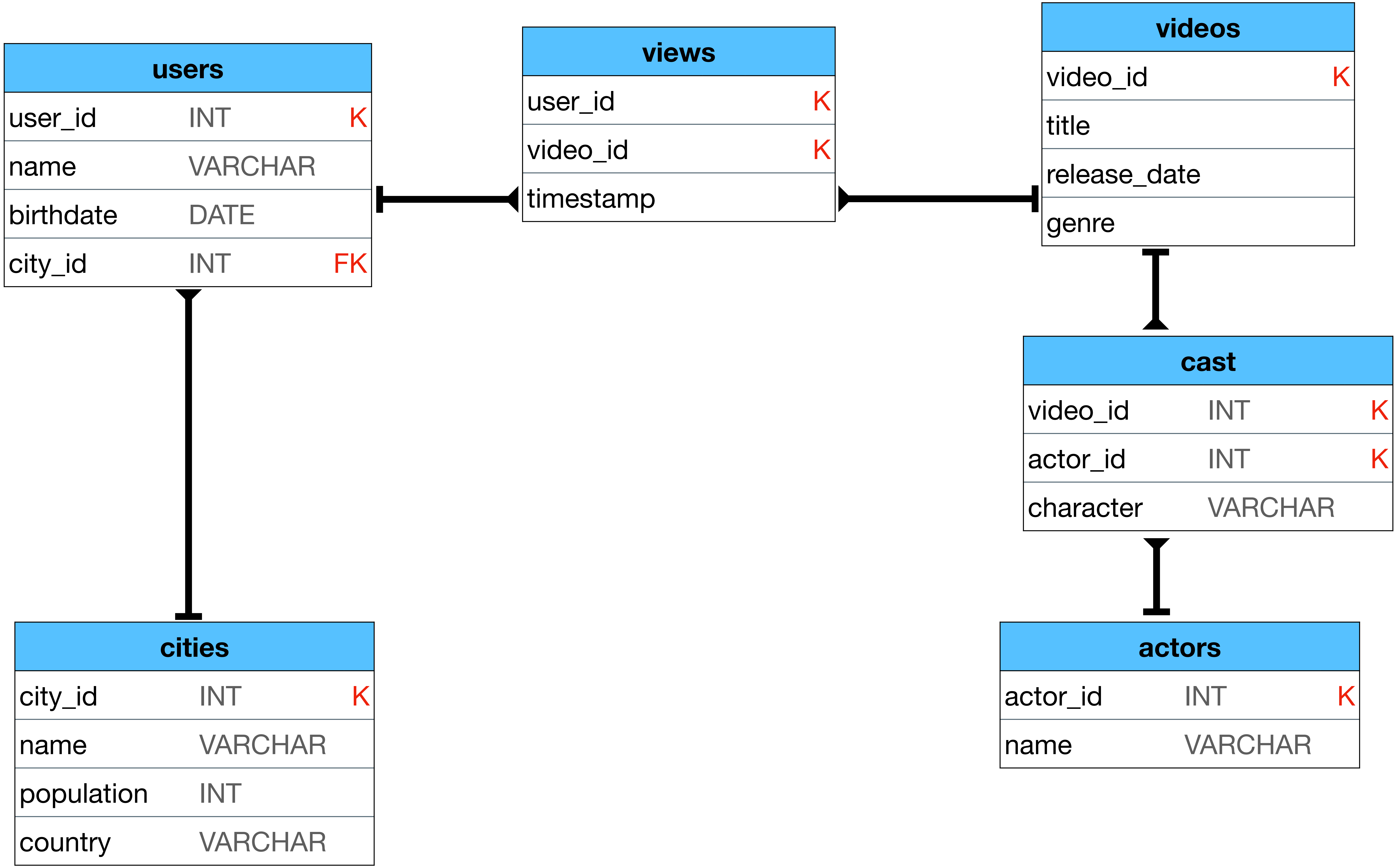
views	
user_id	K
video_id	K
timestamp	

videos	
video_id	K
title	
release_date	
genre	

cities		
city_id	INT	K
name	VARCHAR	
population	INT	
country	VARCHAR	

cast	
video_id	K
actor_id	K
character	

actors		
actor_id	INT	K
name	VARCHAR	



users		
user_id	INT	K
name	VARCHAR	
birthdate	DATE	
city_id	INT	FK

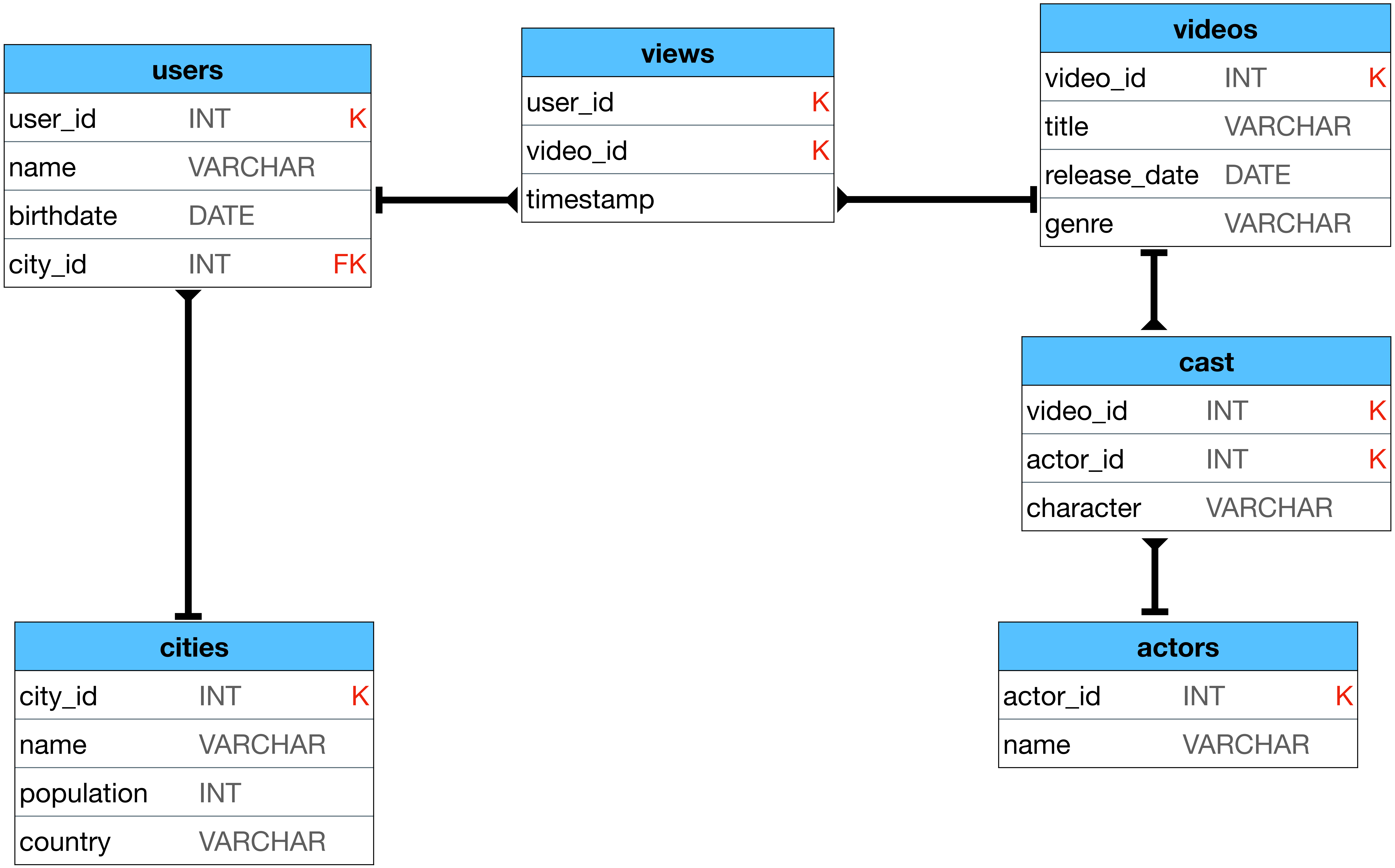
views	
user_id	K
video_id	K
timestamp	

videos	
video_id	K
title	
release_date	
genre	

cities		
city_id	INT	K
name	VARCHAR	
population	INT	
country	VARCHAR	

cast		
video_id	INT	K
actor_id	INT	K
character	VARCHAR	

actors		
actor_id	INT	K
name	VARCHAR	



users		
user_id	INT	K
name	VARCHAR	
birthdate	DATE	
city_id	INT	FK

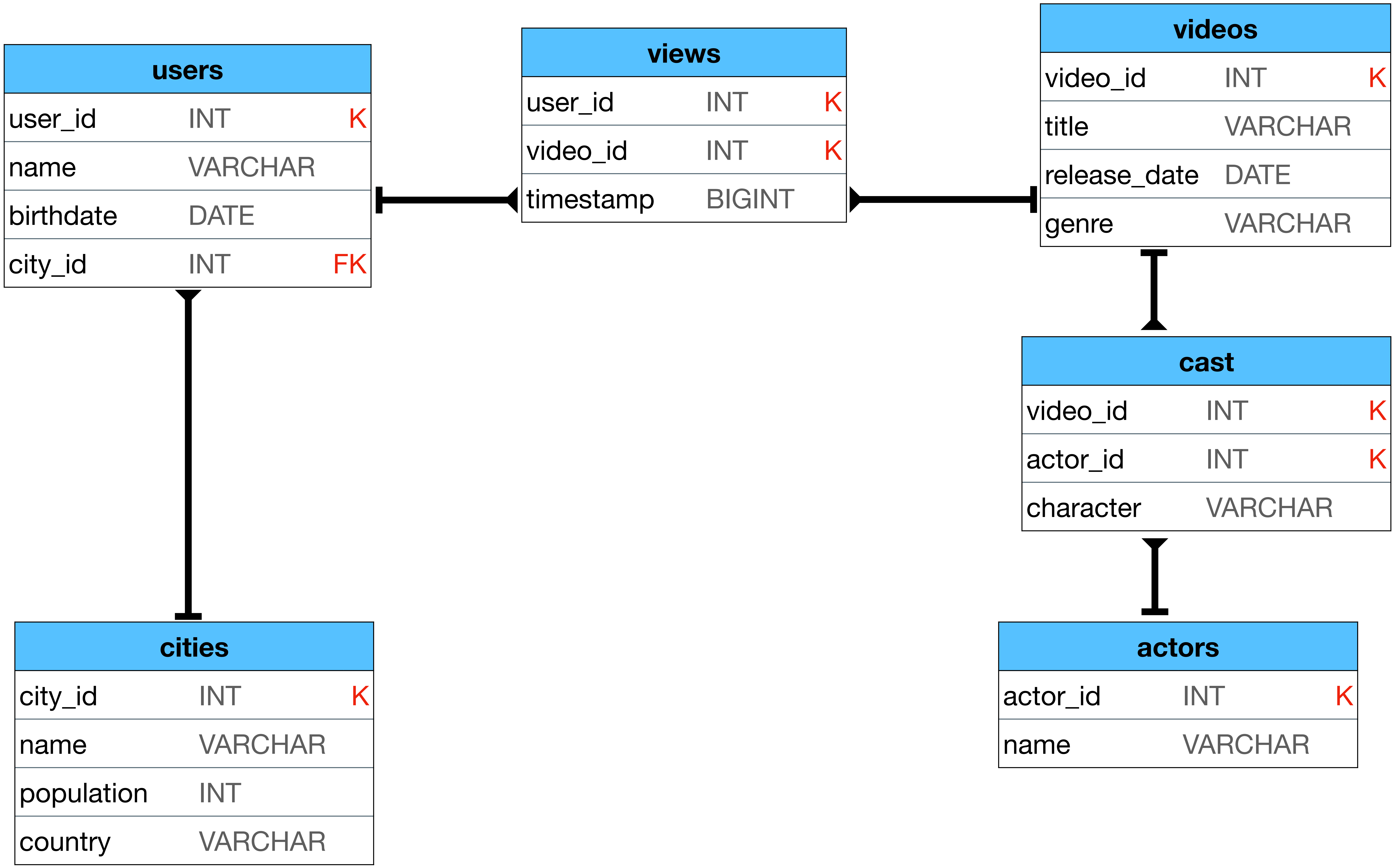
views	
user_id	K
video_id	K
timestamp	

videos		
video_id	INT	K
title	VARCHAR	
release_date	DATE	
genre	VARCHAR	

cast		
video_id	INT	K
actor_id	INT	K
character	VARCHAR	

cities		
city_id	INT	K
name	VARCHAR	
population	INT	
country	VARCHAR	

actors		
actor_id	INT	K
name	VARCHAR	



users		
user_id	INT	K
name	VARCHAR	
birthdate	DATE	
city_id	INT	FK

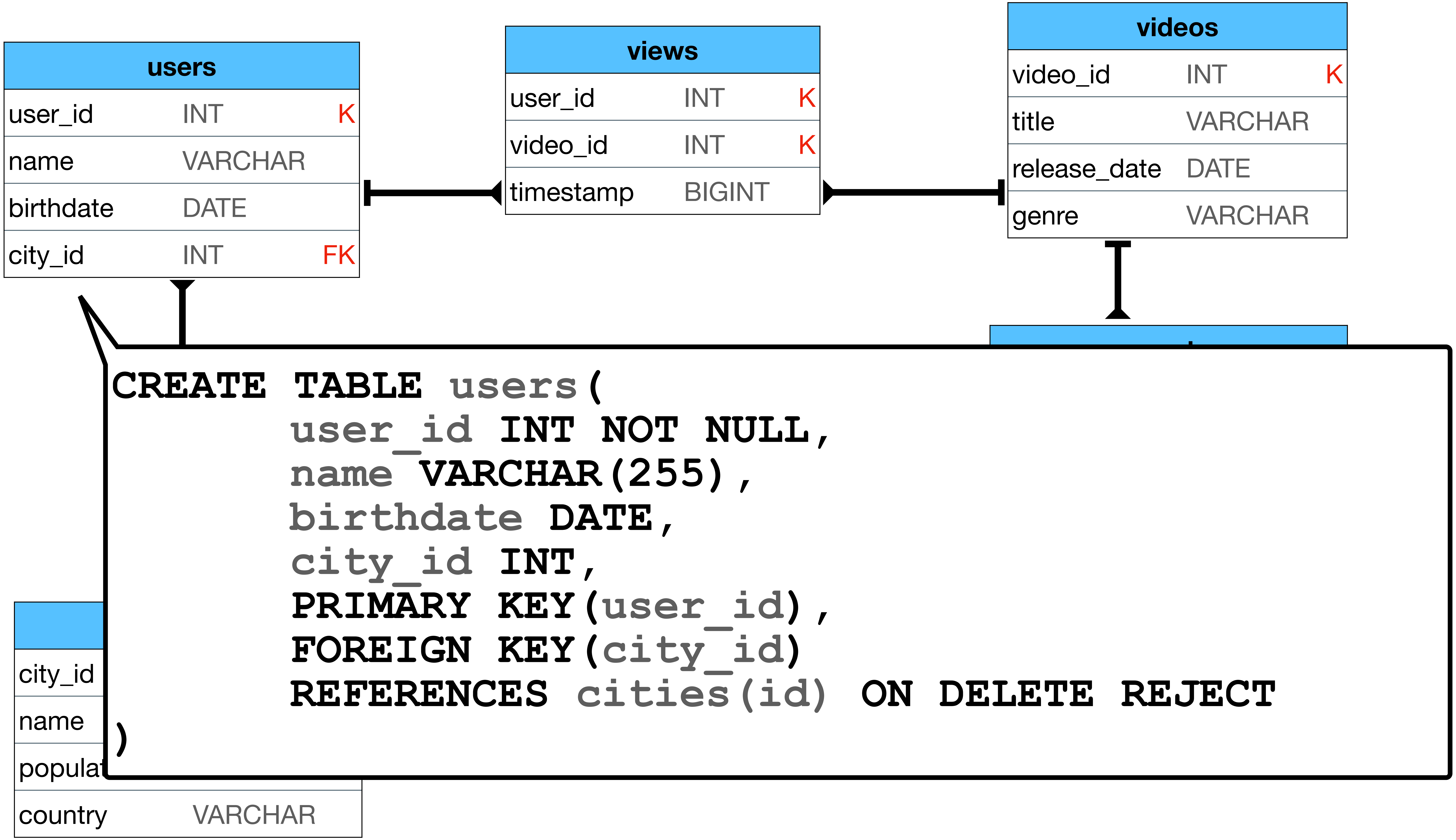
views		
user_id	INT	K
video_id	INT	K
timestamp	BIGINT	

videos		
video_id	INT	K
title	VARCHAR	
release_date	DATE	
genre	VARCHAR	

cast		
video_id	INT	K
actor_id	INT	K
character	VARCHAR	

cities		
city_id	INT	K
name	VARCHAR	
population	INT	
country	VARCHAR	

actors		
actor_id	INT	K
name	VARCHAR	



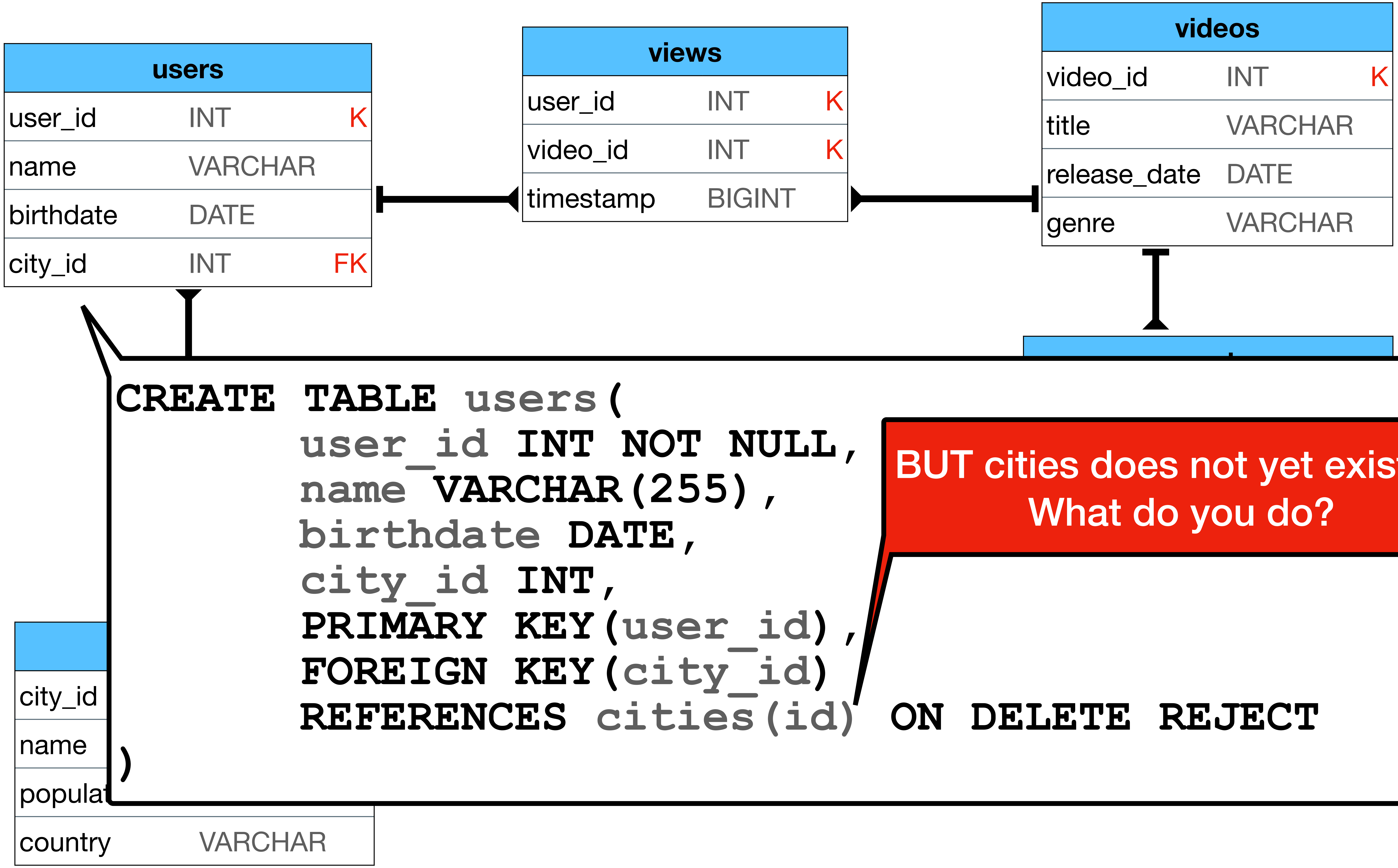
users		
user_id	INT	K
name	VARCHAR	
birthdate	DATE	
city_id	INT	FK

views		
user_id	INT	K
video_id	INT	K
timestamp	BIGINT	

videos		
video_id	INT	K
title	VARCHAR	
release_date	DATE	
genre	VARCHAR	

city_id	
name	
populat	
country	VARCHAR

```
CREATE TABLE users(
        user_id INT NOT NULL,
        name VARCHAR(255),
        birthdate DATE,
        city_id INT,
        PRIMARY KEY(user_id),
        FOREIGN KEY(city_id)
        REFERENCES cities(id) ON DELETE REJECT
)
```



users		
user_id	INT	K
name	VARCHAR	
birthdate	DATE	
city_id	INT	FK

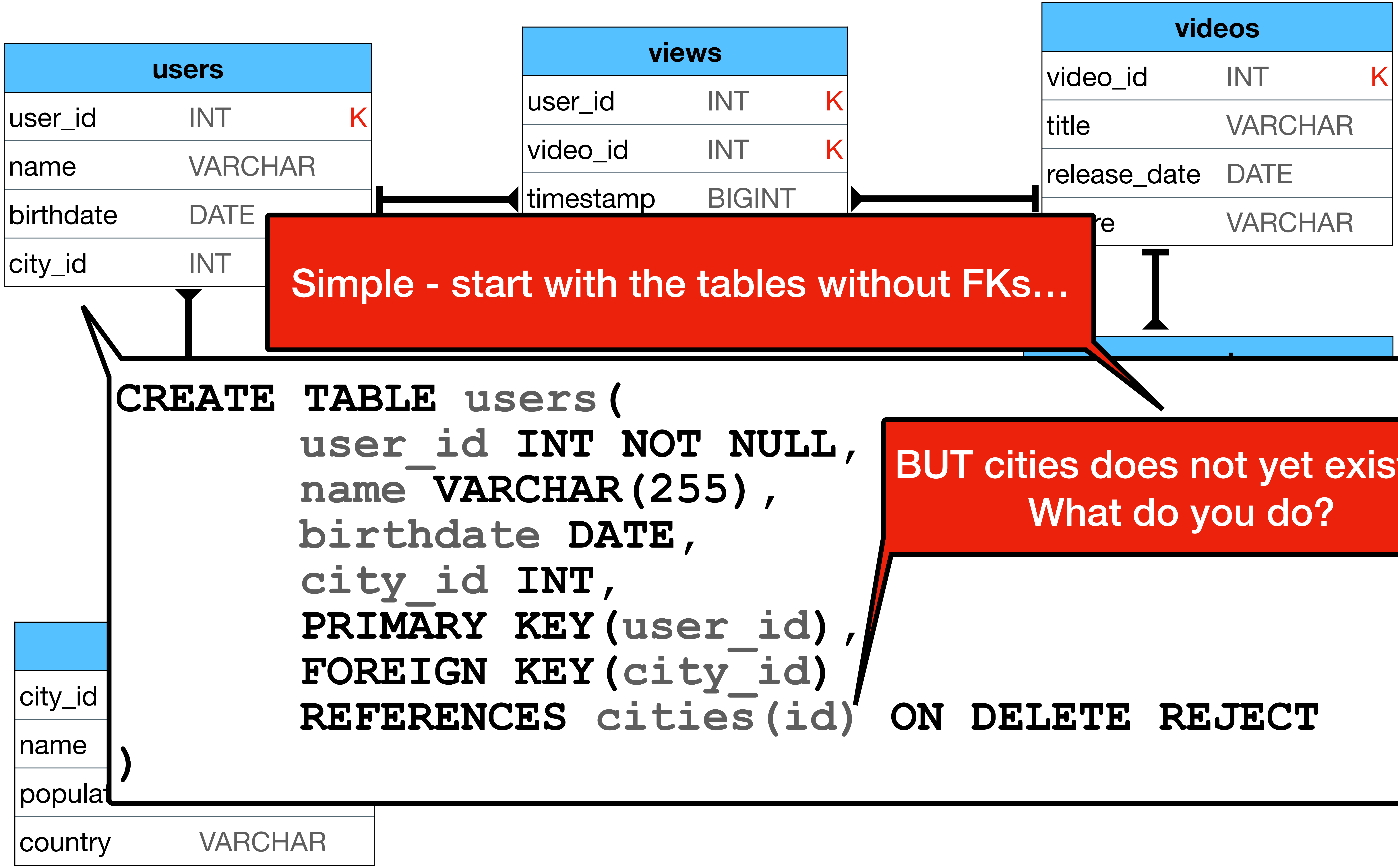
views		
user_id	INT	K
video_id	INT	K
timestamp	BIGINT	

videos		
video_id	INT	K
title	VARCHAR	
release_date	DATE	
genre	VARCHAR	

city_id	
name	
populat	
country	VARCHAR

```
CREATE TABLE users(
        user_id INT NOT NULL,
        name VARCHAR(255),
        birthdate DATE,
        city_id INT,
        PRIMARY KEY(user_id),
        FOREIGN KEY(city_id)
        REFERENCES cities(id) ON DELETE REJECT
)
```

BUT cities does not yet exists…
What do you do?



Simple - start with the tables without FKs…

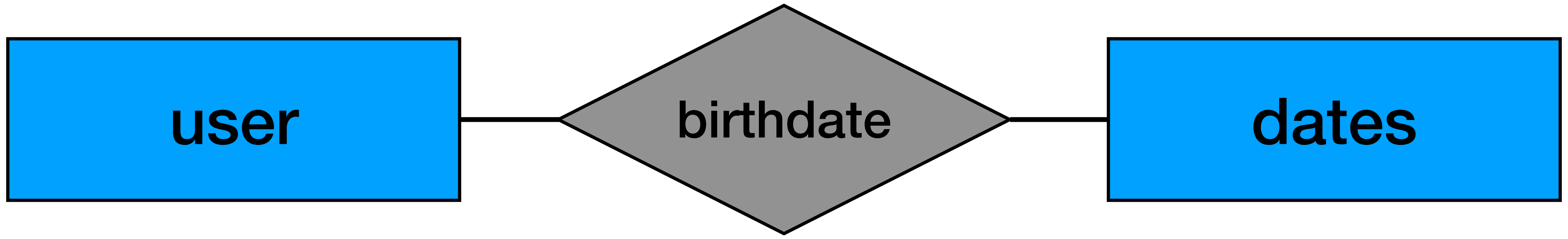
```
CREATE TABLE users(
       user_id INT NOT NULL,
       name VARCHAR(255),
       birthdate DATE,
       city_id INT,
       PRIMARY KEY(user_id),
       FOREIGN KEY(city_id)
       REFERENCES cities(id) ON DELETE REJECT
)
```

BUT cities does not yet exists…
What do you do?

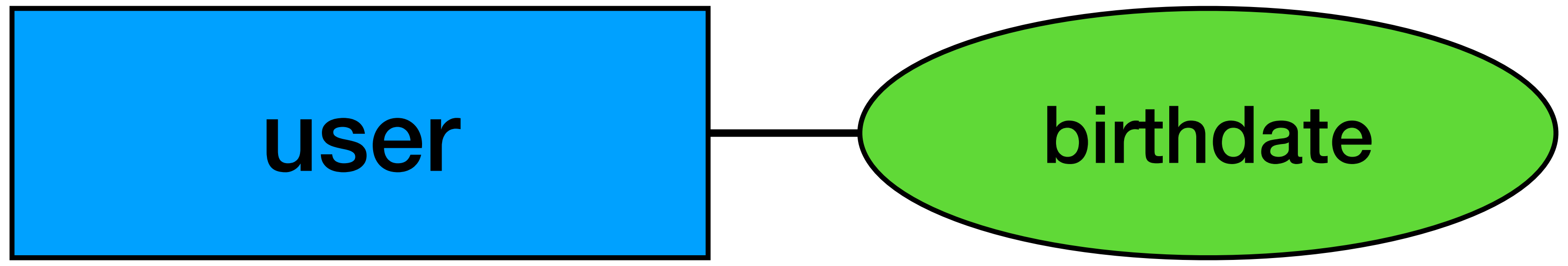
Design examples

Example (1)

- What is the problem here? Solution?

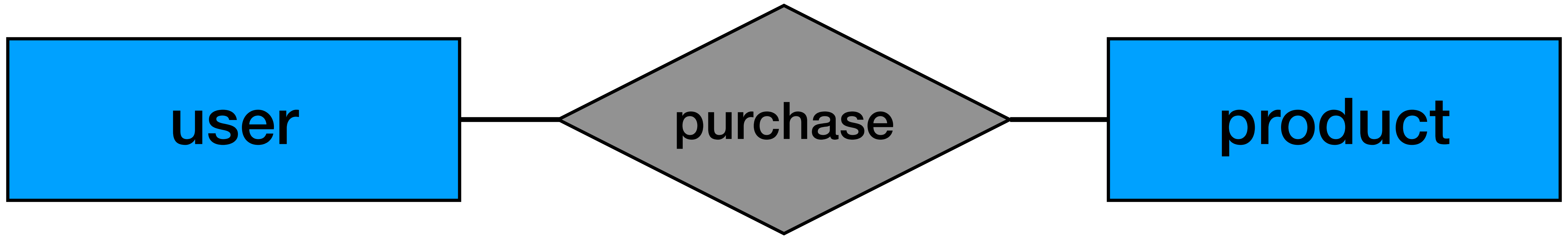


Example (1)

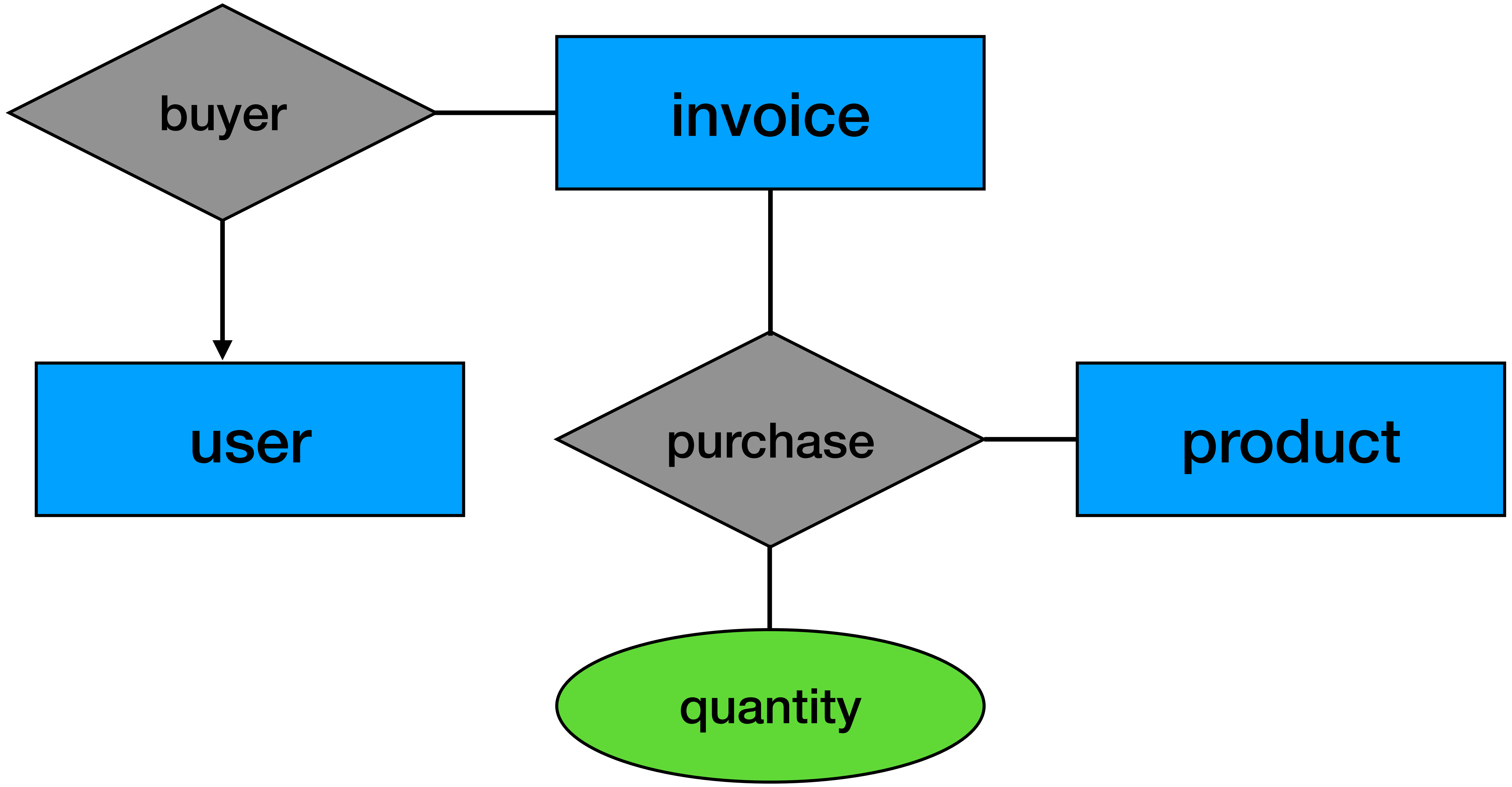


Example (2)

- What is the problem here? Solution?

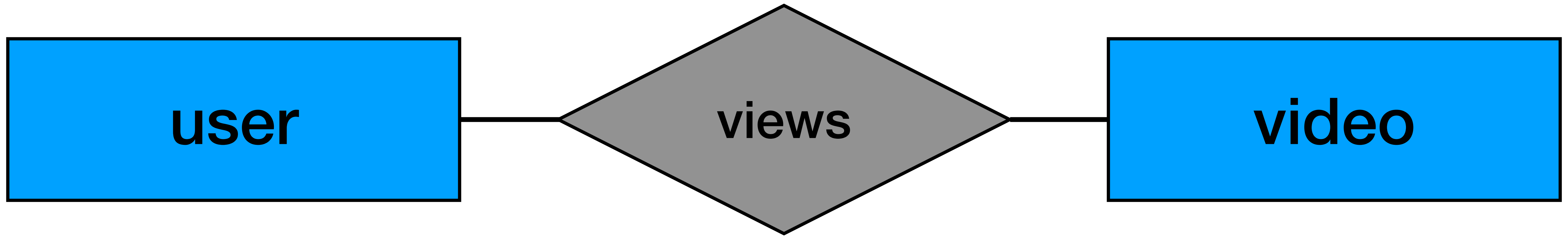


Example (2)



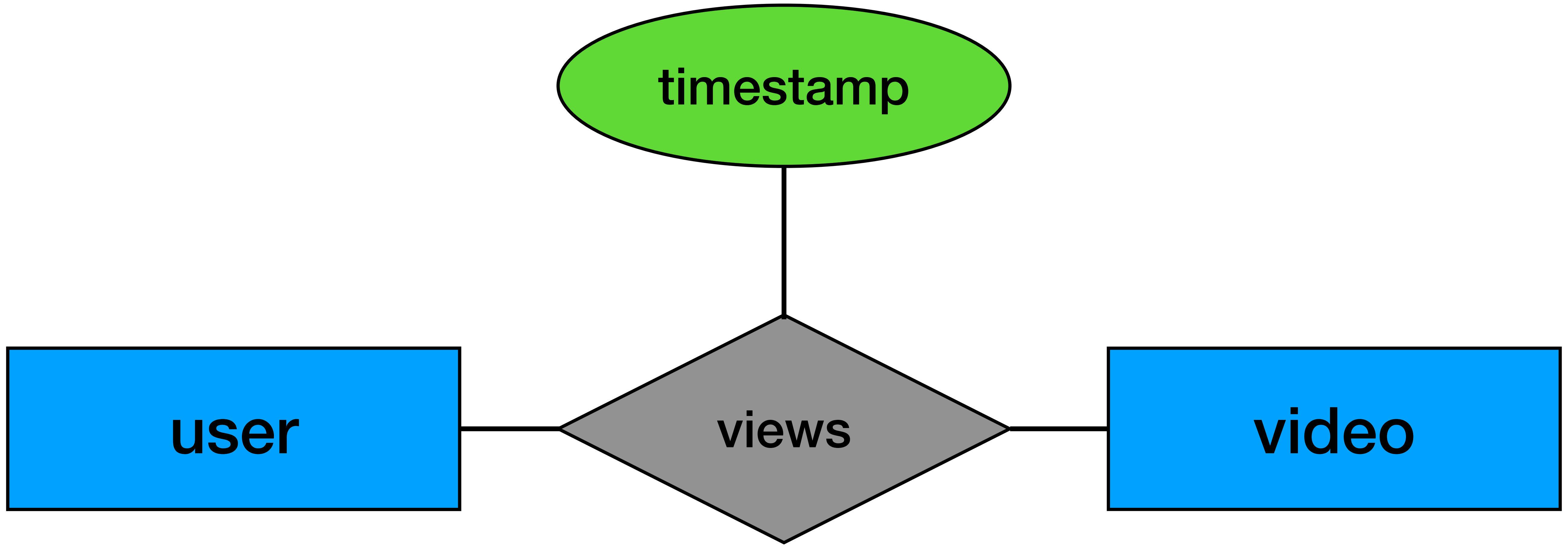
Example (3)

- What is the problem here? Solution?



Example (3)

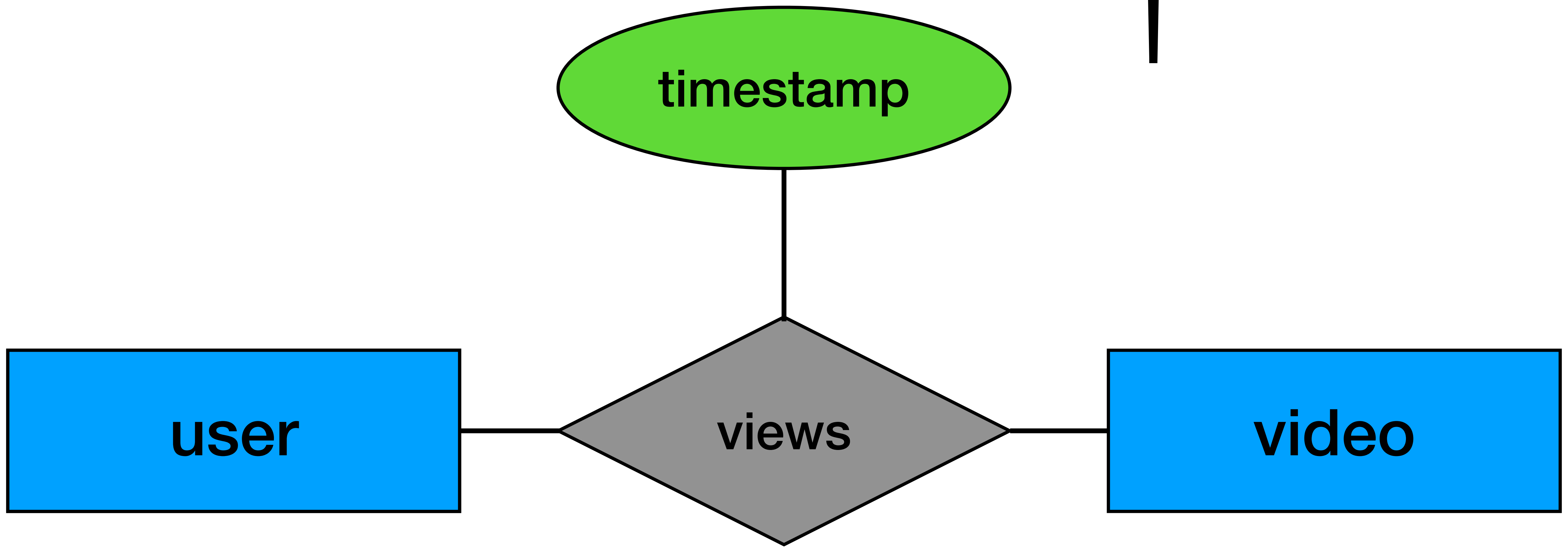
- Option 1



Example (3)

- Option 1

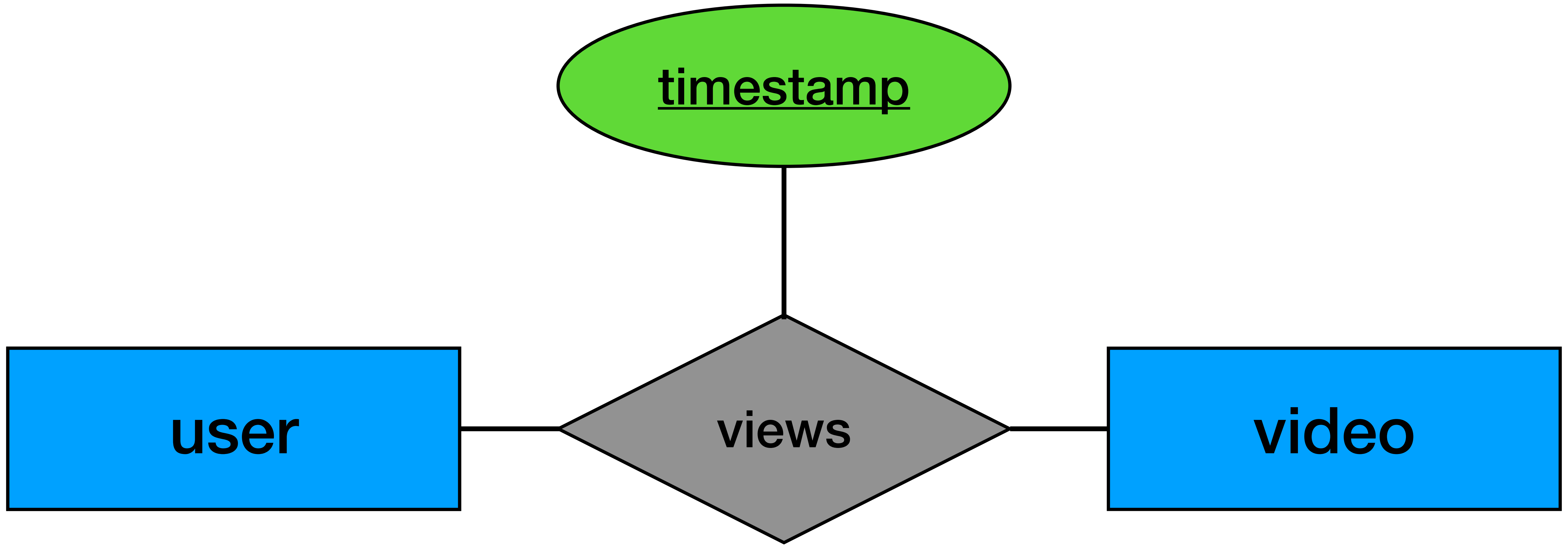
Is this ok?



views_option_1		
user_id	INT	K
video_id	INT	K
timestamp	BIGINT	

Example (3)

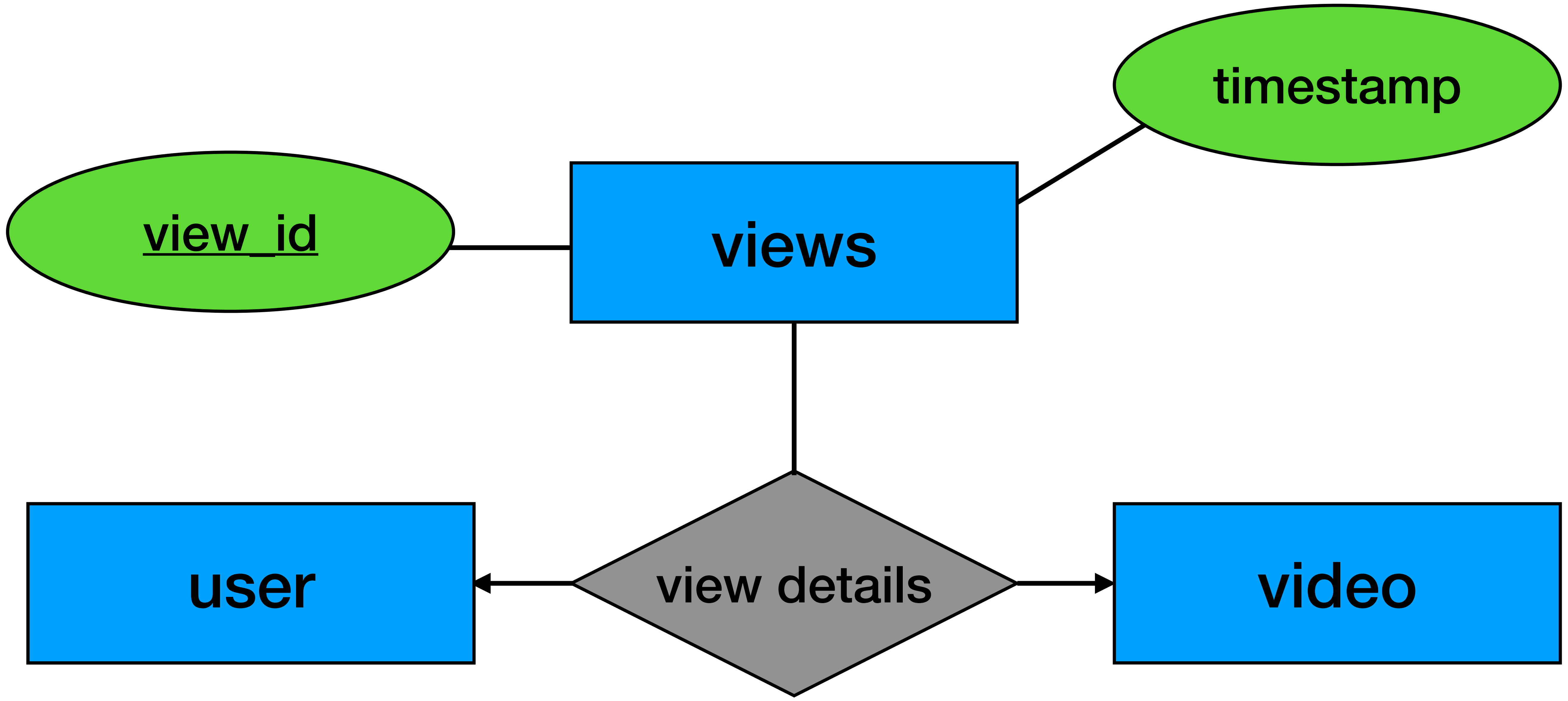
- Option 1



views_option_1		
user_id	INT	K
video_id	INT	K
timestamp	BIGINT	K

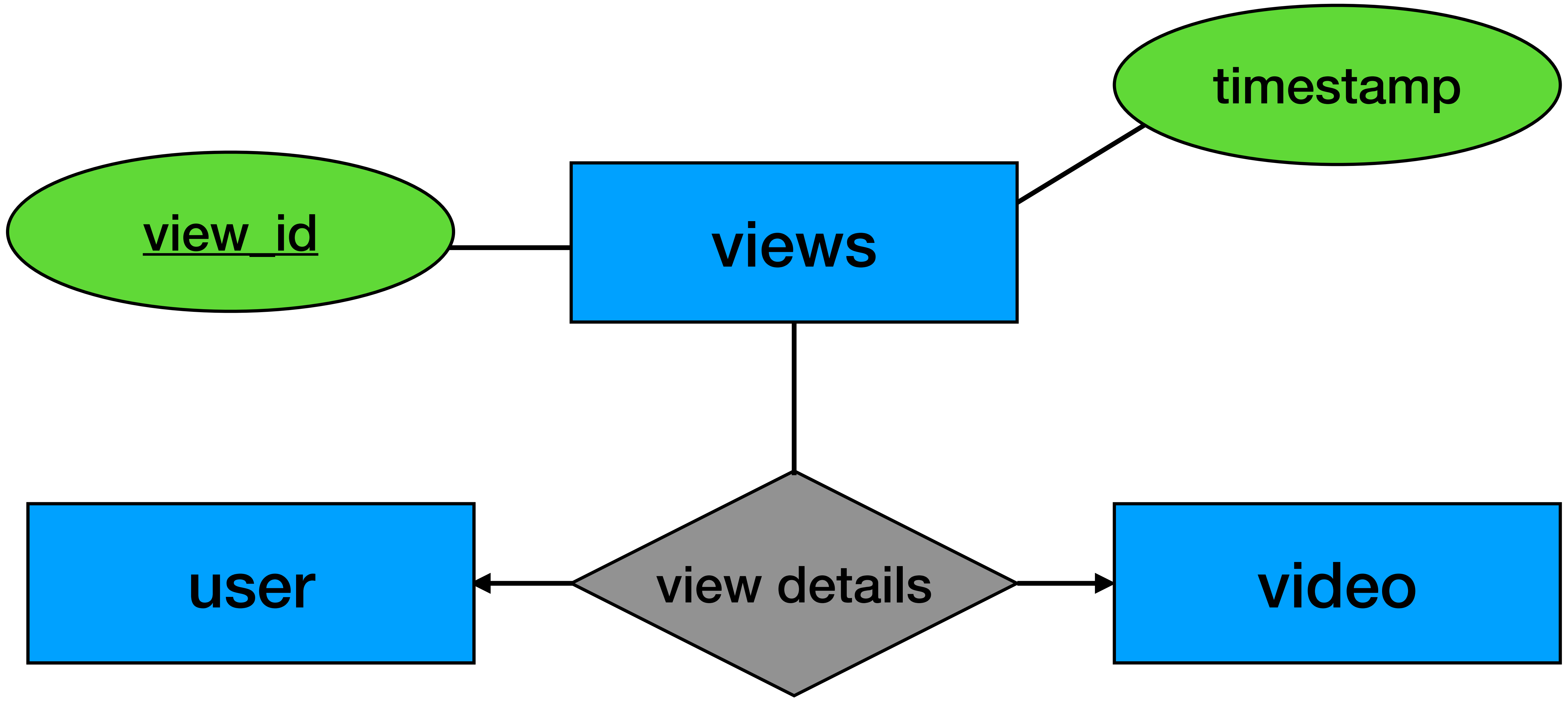
Example (3)

- Option 2



Example (3)

- Option 2



views_option_2		
view_id	INT	K
user_id	INT	FK
video_id	INT	FK
timestamp	BIGINT	

Example (3)

- Option 1 vs Option 2

views_option_1		
user_id	INT	K
video_id	INT	K
timestamp	BIGINT	K

Classic relational modeling -
“By the book”

views_option_2		
view_id	INT	K
user_id	INT	FK
video_id	INT	FK
timestamp	BIGINT	

“NoSQL style” -
Can improve performance on large scale

Exa

- Op

Open discussion

Assume the data is **stored on disk** by the **order of the primary key**

Can you think of a query that would be “optimized” for each option?

views_option_1		
user_id	INT	K
video_id	INT	K
timestamp	BIGINT	K

Classic relational modeling -
“By the book”

views_option_2		
view_id	INT	K
user_id	INT	FK
video_id	INT	FK
timestamp	BIGINT	

“NoSQL style” -
Can improve performance on large scale

Exa

- Op

Open discussion

Assume the data is **stored on disk** by the **order of the primary key**

Can you think of a query that would be “optimized” for each option?

Return all videos viewed by a user

views_option_1		
user_id	INT	K
video_id	INT	K
timestamp	BIGINT	K

Classic relational modeling - “By the book”

views_option_2		
view_id	INT	K
user_id	INT	FK
video_id	INT	FK
timestamp	BIGINT	

Return all videos viewed last week

“NoSQL style” - Can improve performance on large scale

Exa

- Op

Open discussion

Assume the data is **stored on disk** by the **order of the primary key**

Can you think of a query that would be "optimized" for each option?

Return all videos viewed by a user

views_option_1		
user_id	INT	K
video_id	INT	K
timestamp	BIGINT	K

views_option_2		
view_id	INT	K
user_id	INT	FK
video_id	INT	FK
timestamp	BIGINT	

Return all videos viewed last week

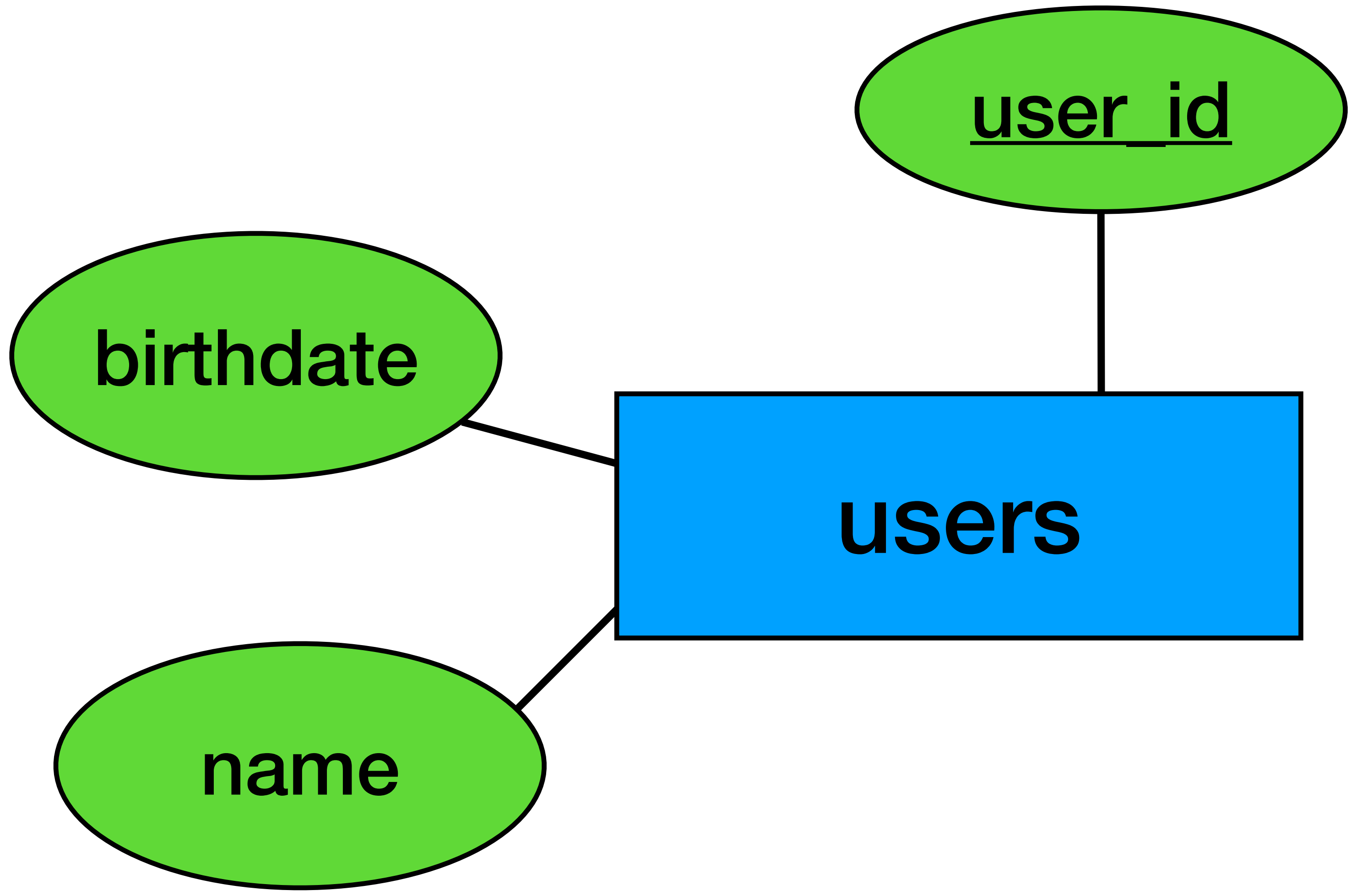
Modeling is an art…
There is not alway a clear right / wrong answer

Classic relational modeling -
"By the book"

"NoSQL style" -
Can improve performance on large scale

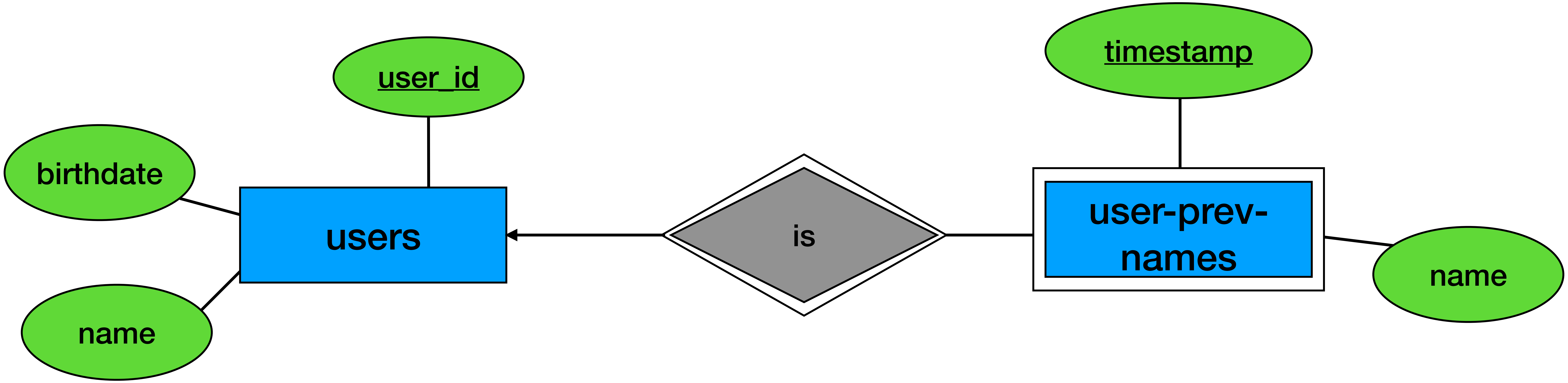
Example (4)

- Add the option to save previous changes to the name attribute



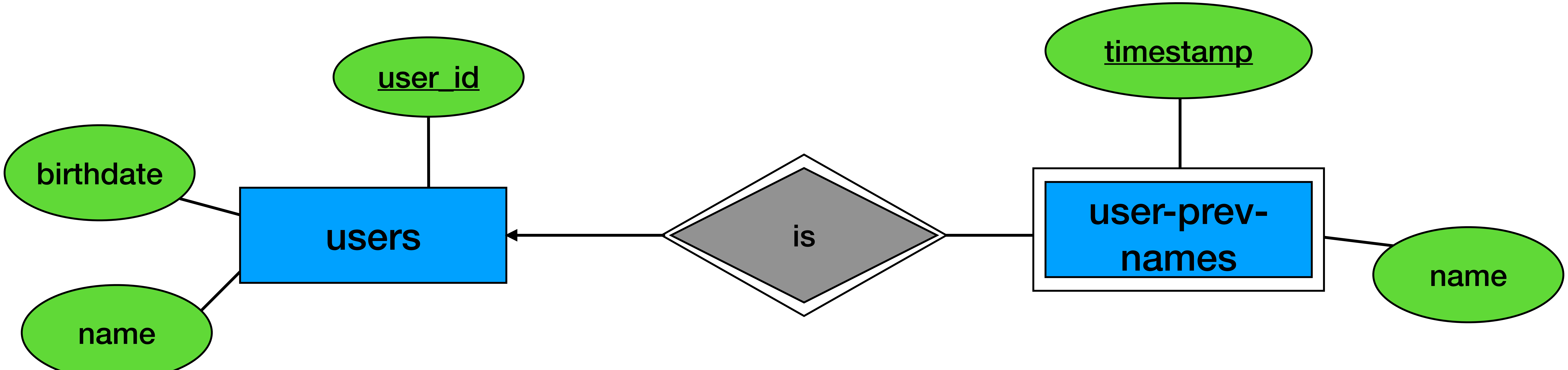
Example (4)

- Add the option to save previous changes to the name attribute



Example (4)

- Add the option to save previous changes to the name attribute

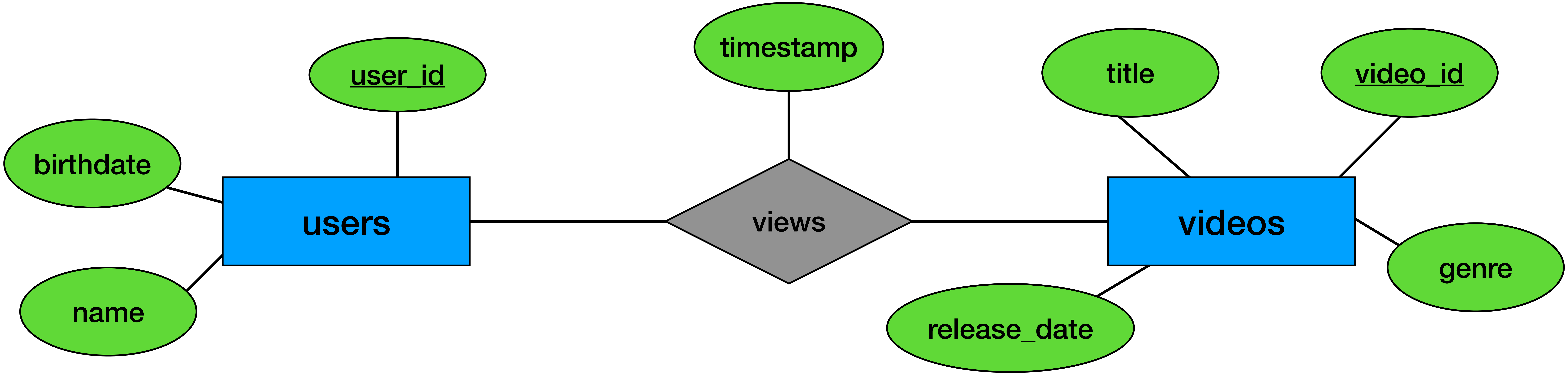


users	
user_id	K
name	
birthdate	

user_names	
user_id	K
timestamp	K
name	

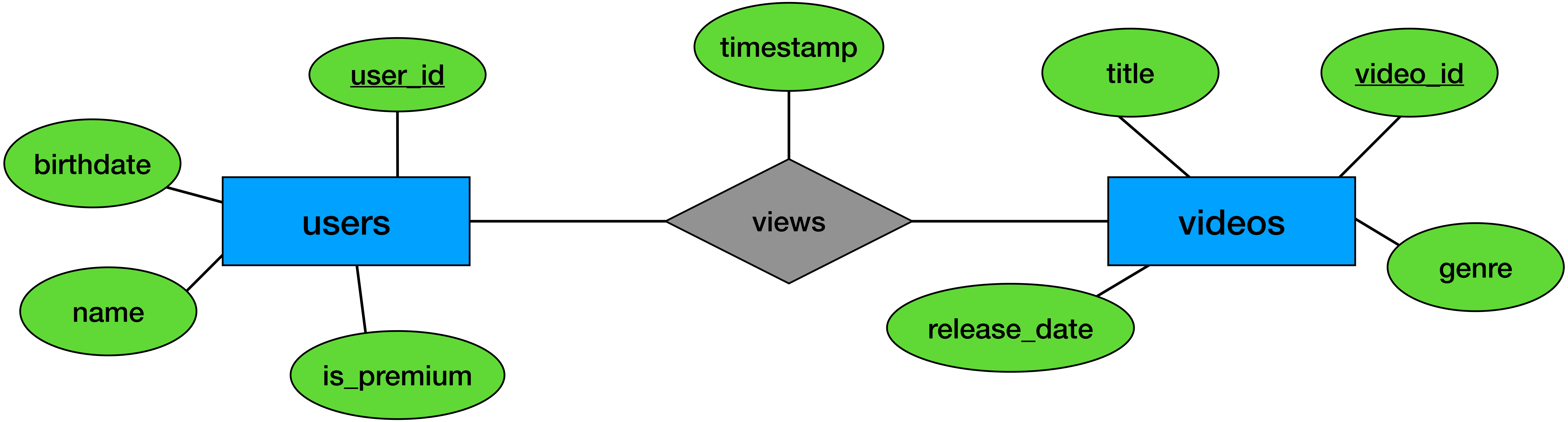
Example (5)

- Add the option for a “premium” user



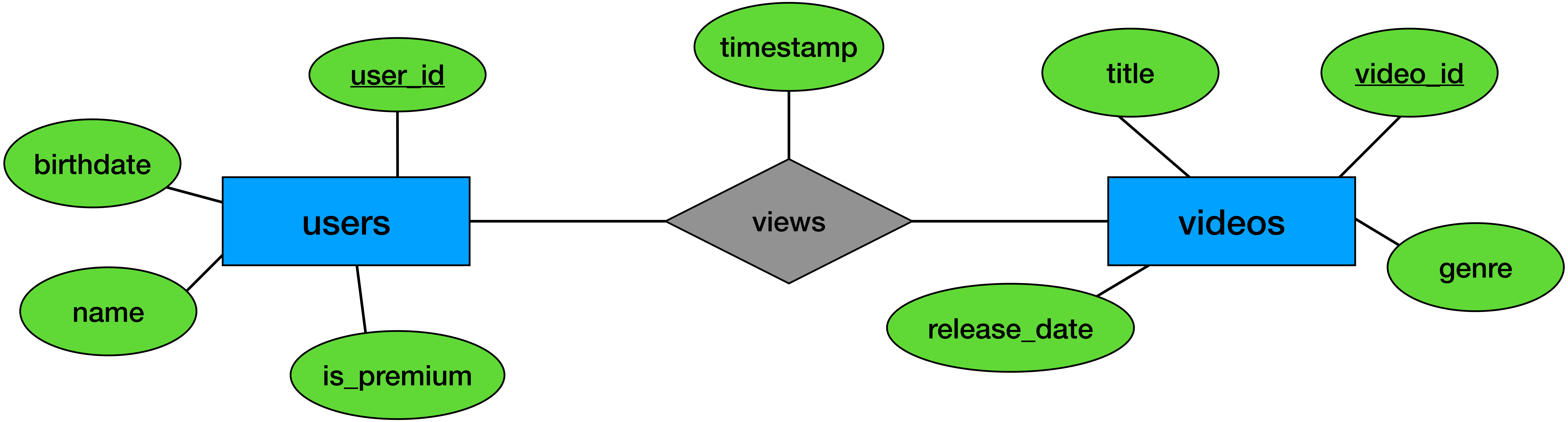
Example (5)

- Add the option for a "premium" user



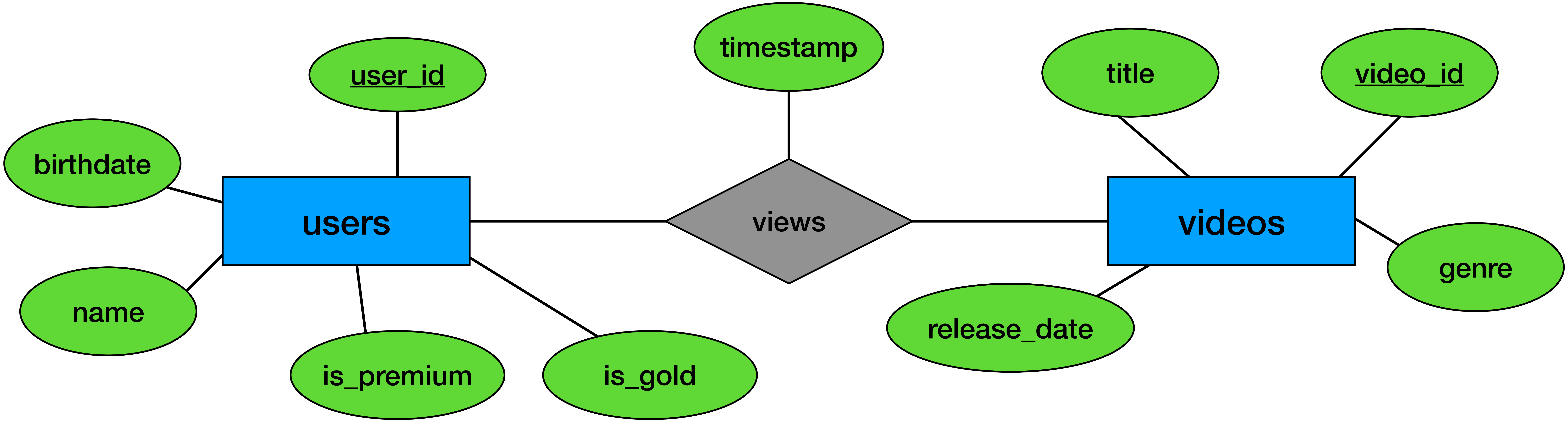
Example (5)

- Add the option for a “premium” user or “gold” user



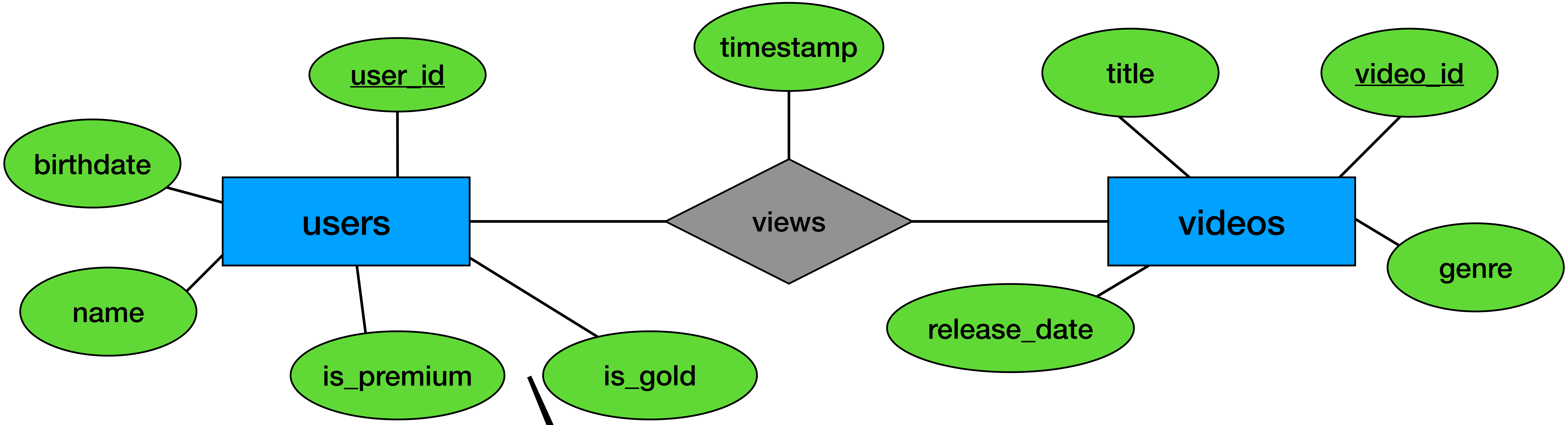
Example (5)

- Add the option for a “premium” user or “gold” user



Example (5)

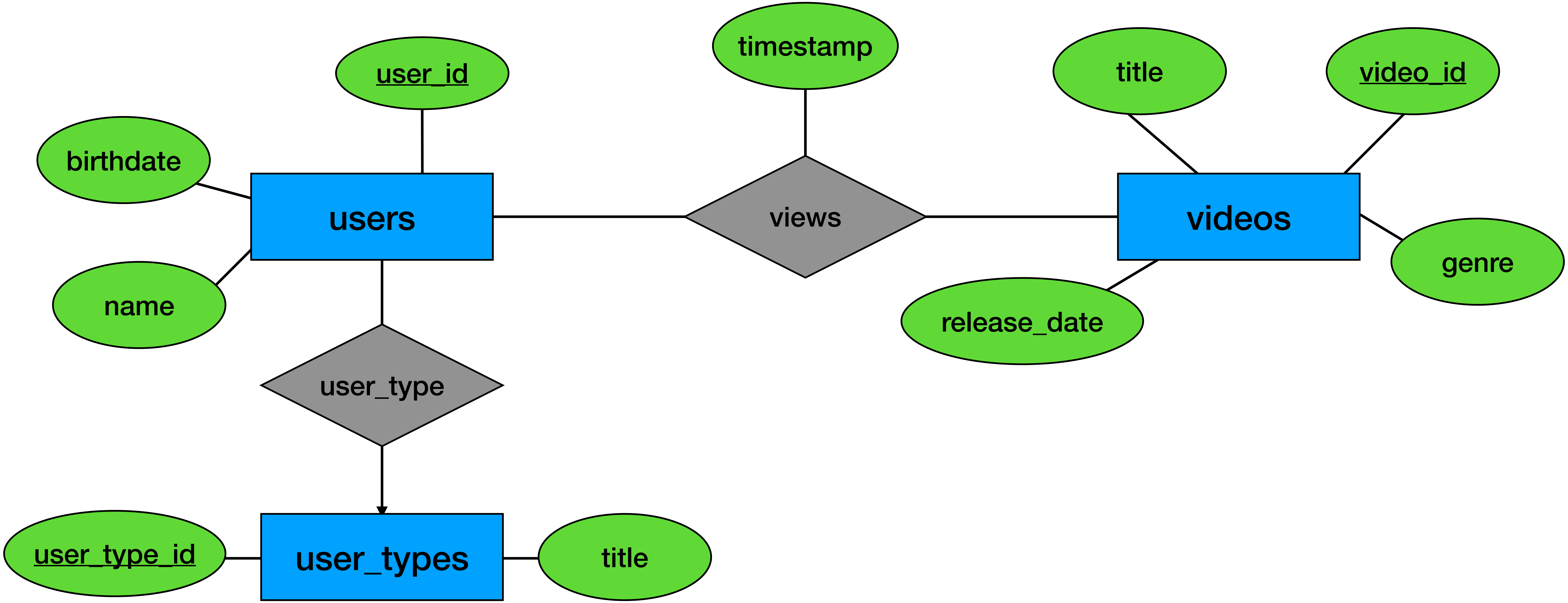
- Add the option for a “premium” user or “gold” user



Is this ok?

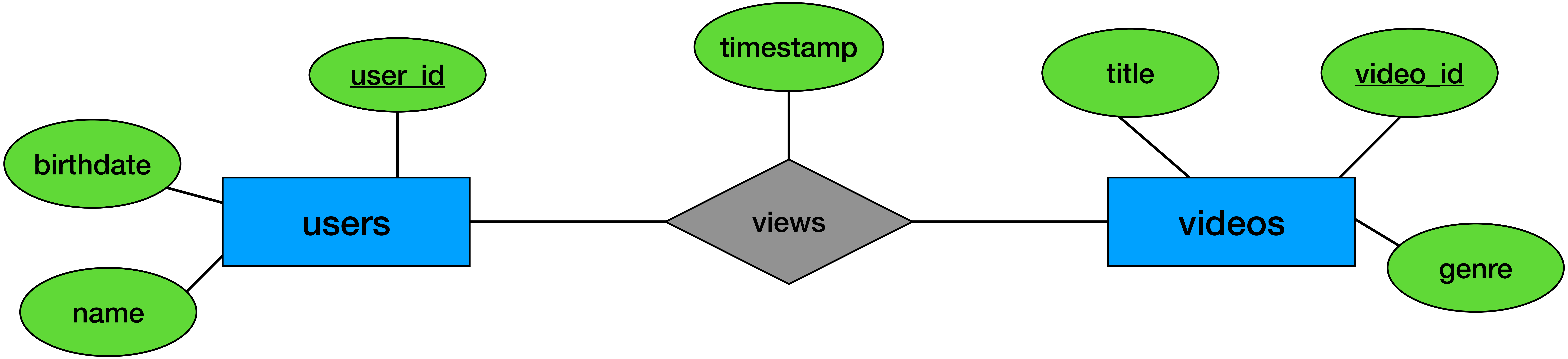
Example (5)

- Add the option for a “premium” user or “gold” user



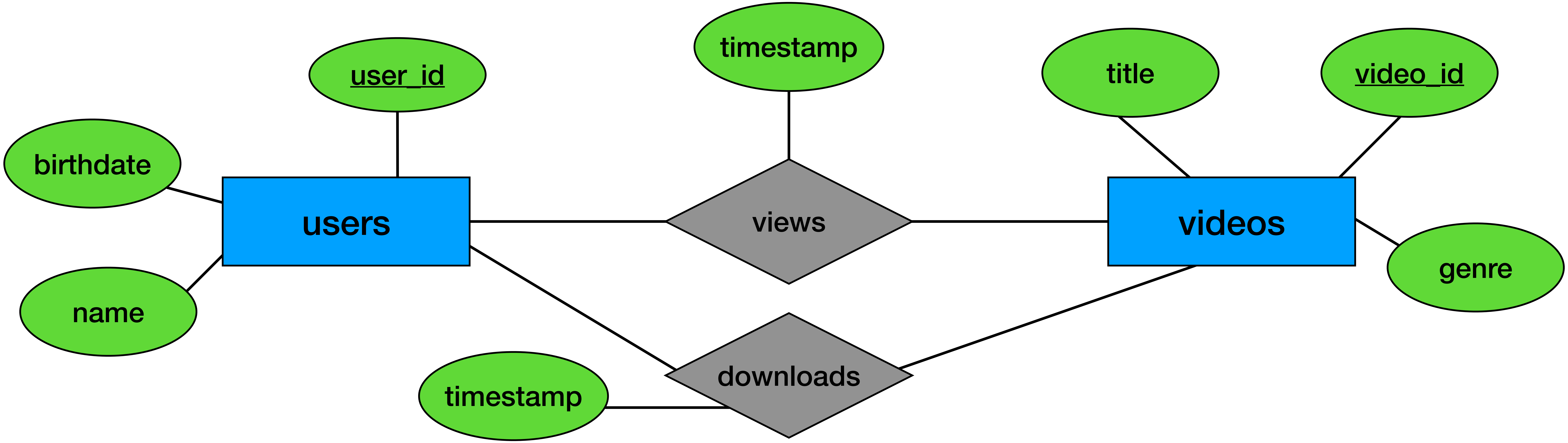
Example (6)

- Add the option to “download” videos



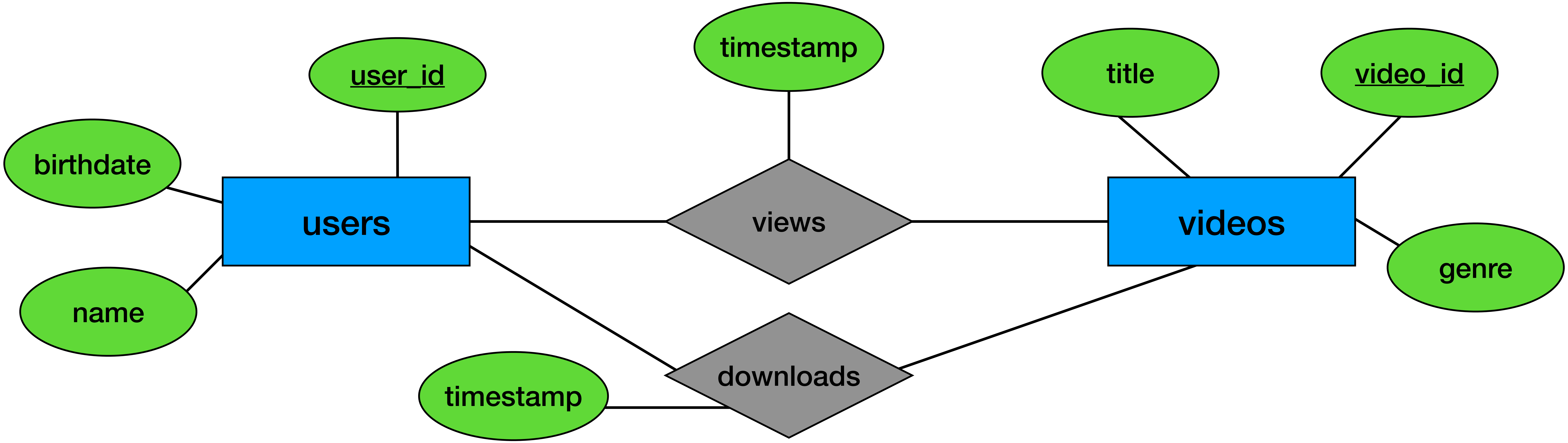
Example (6)

- Add the option to “download” videos



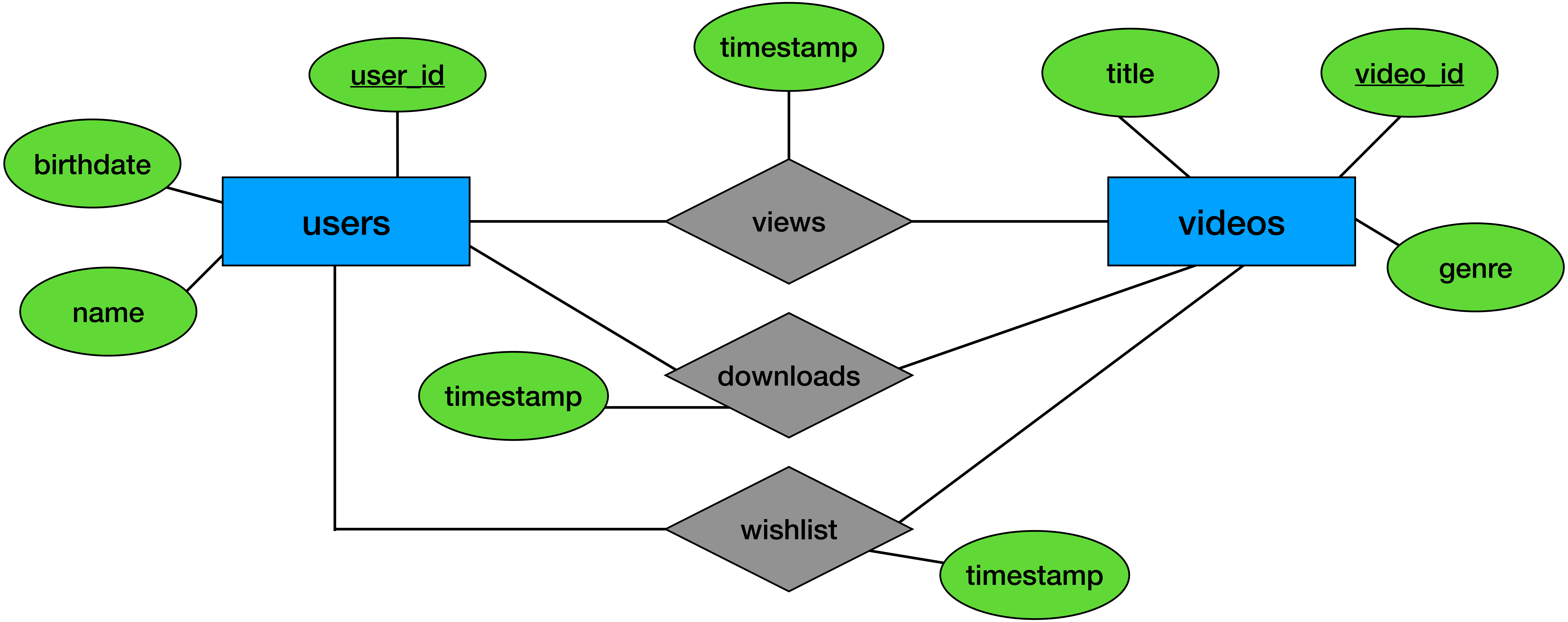
Example (6)

- Add also the option for “wish list”



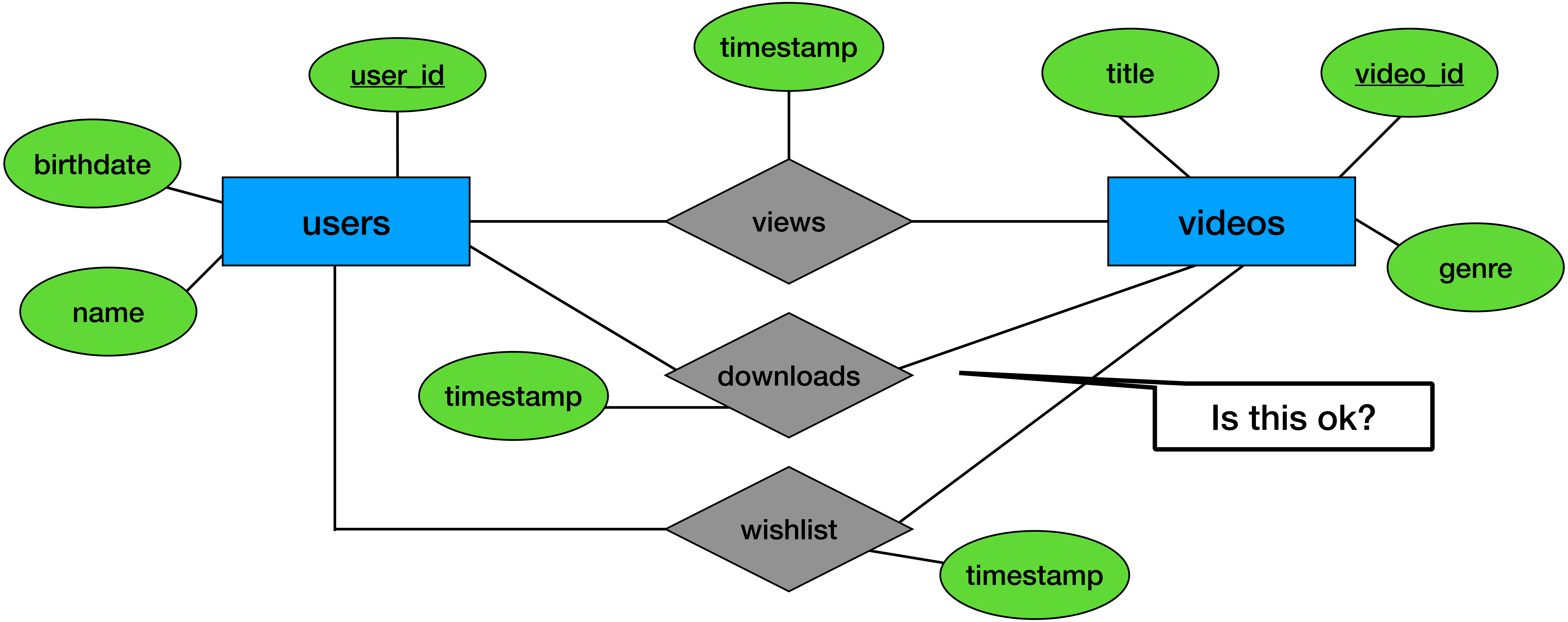
Example (6)

- Add also the option for “wish list”



Example (6)

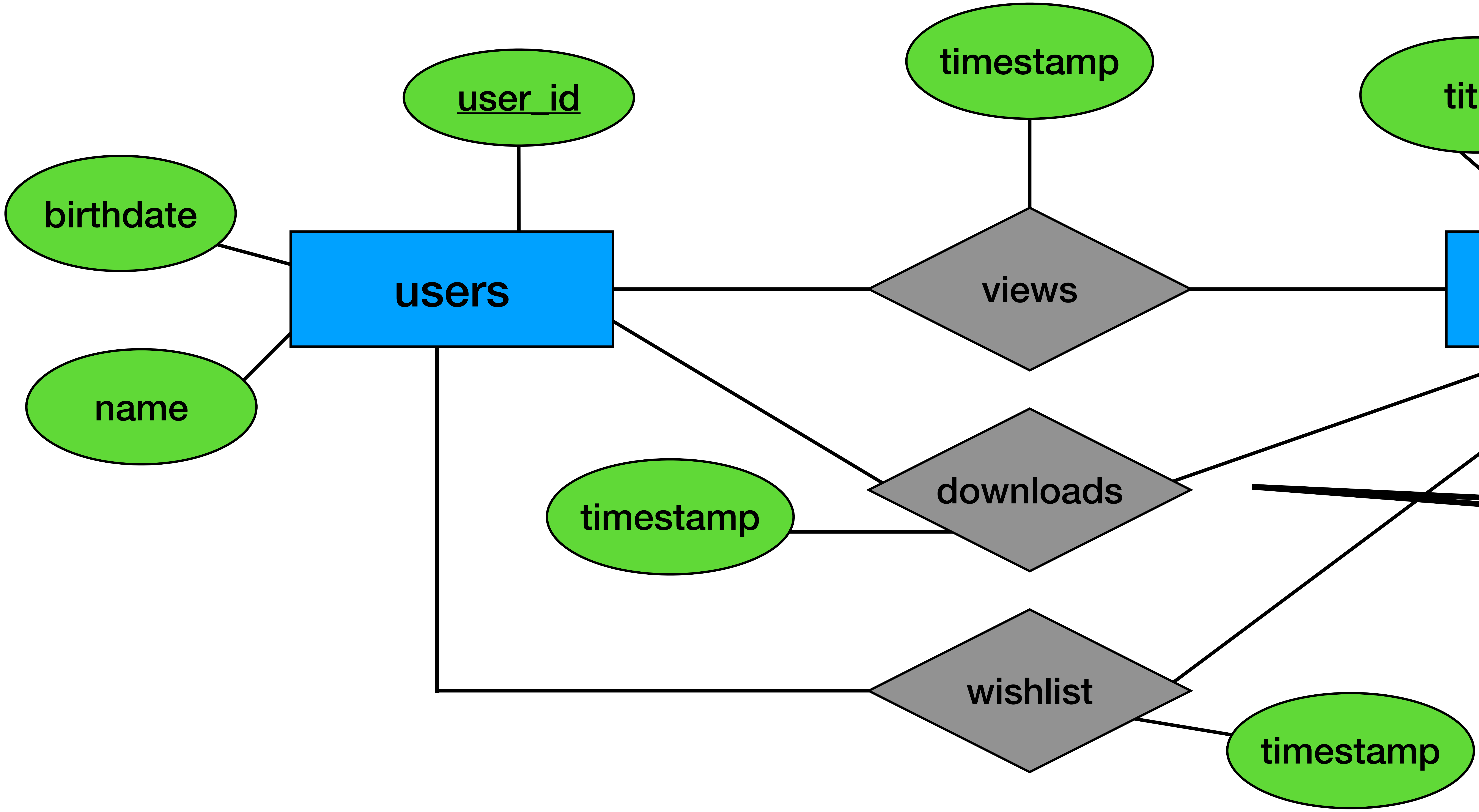
- Add also the option for “wish list”



Is this ok?

Example (6)

- Add also the option for “wish list”

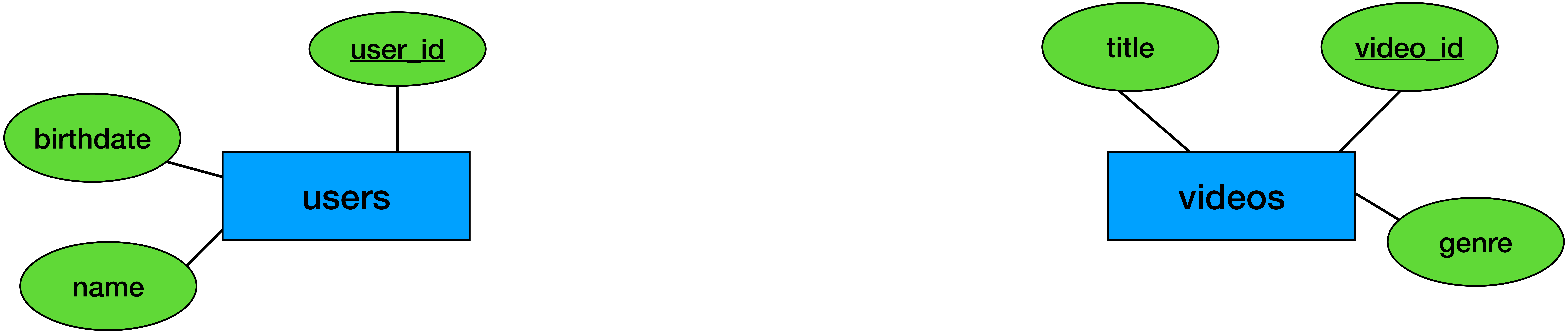


Is this ok?

Yes, but can you think about another way?

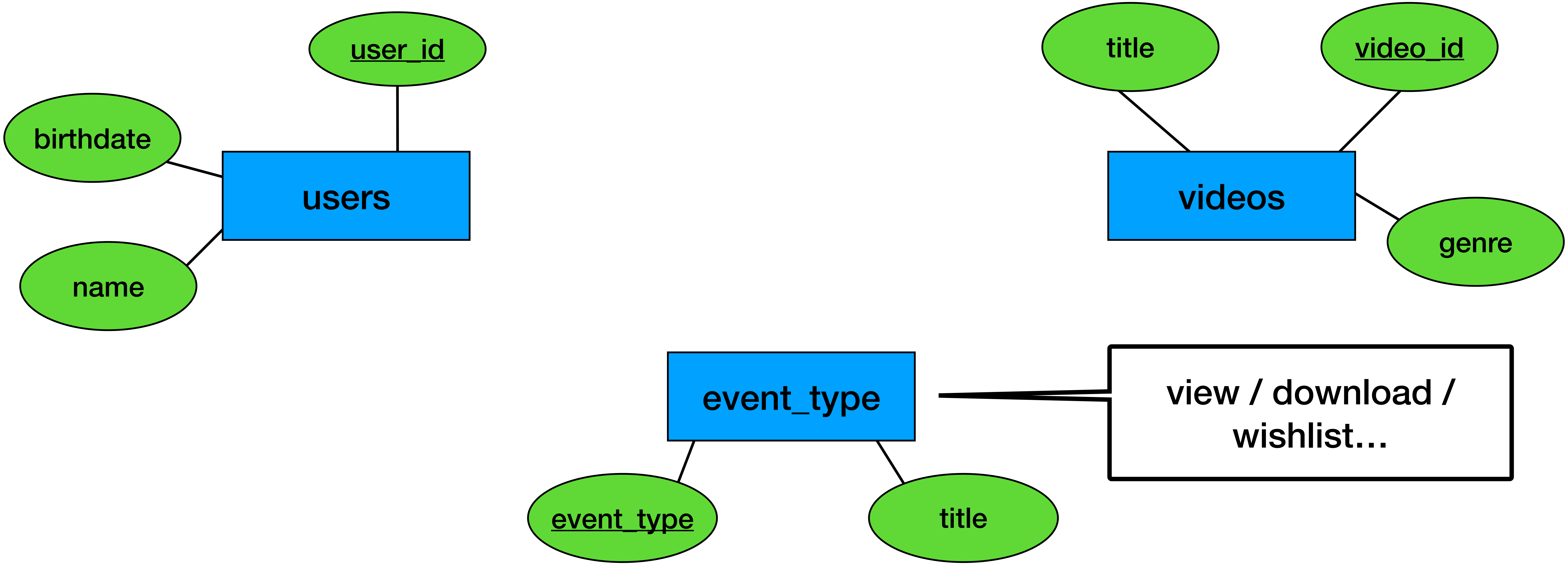
Example (6)

- Convert to “events"



Example (6)

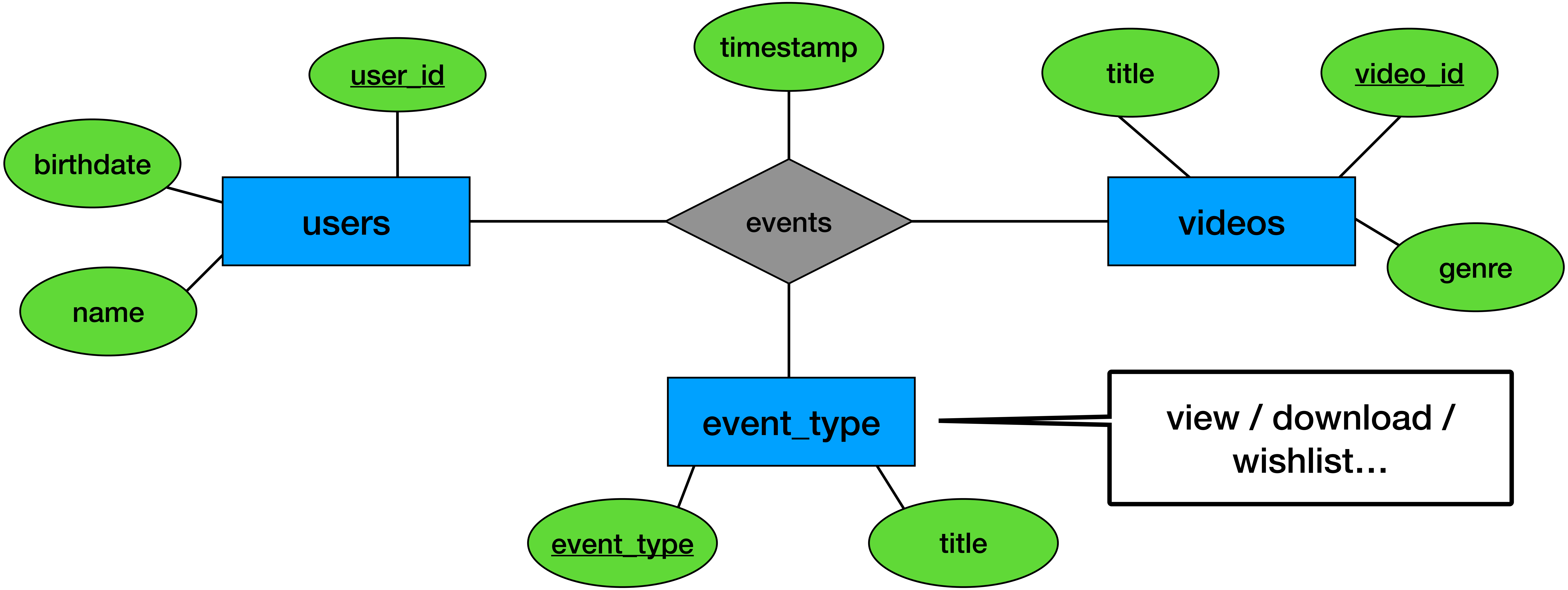
- Convert to “events"



view / download / wishlist...

Example (6)

- Convert to “events"

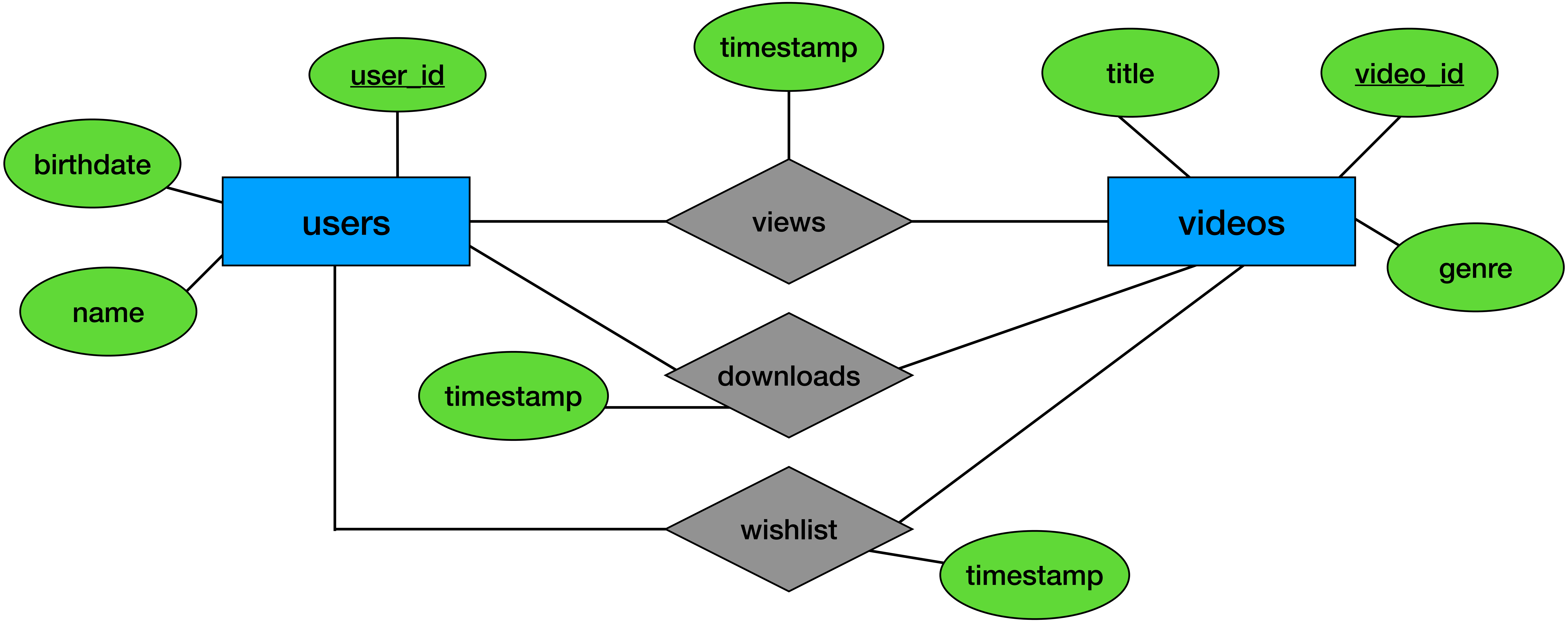


Example (6)

- How would the tables look like for both versions?

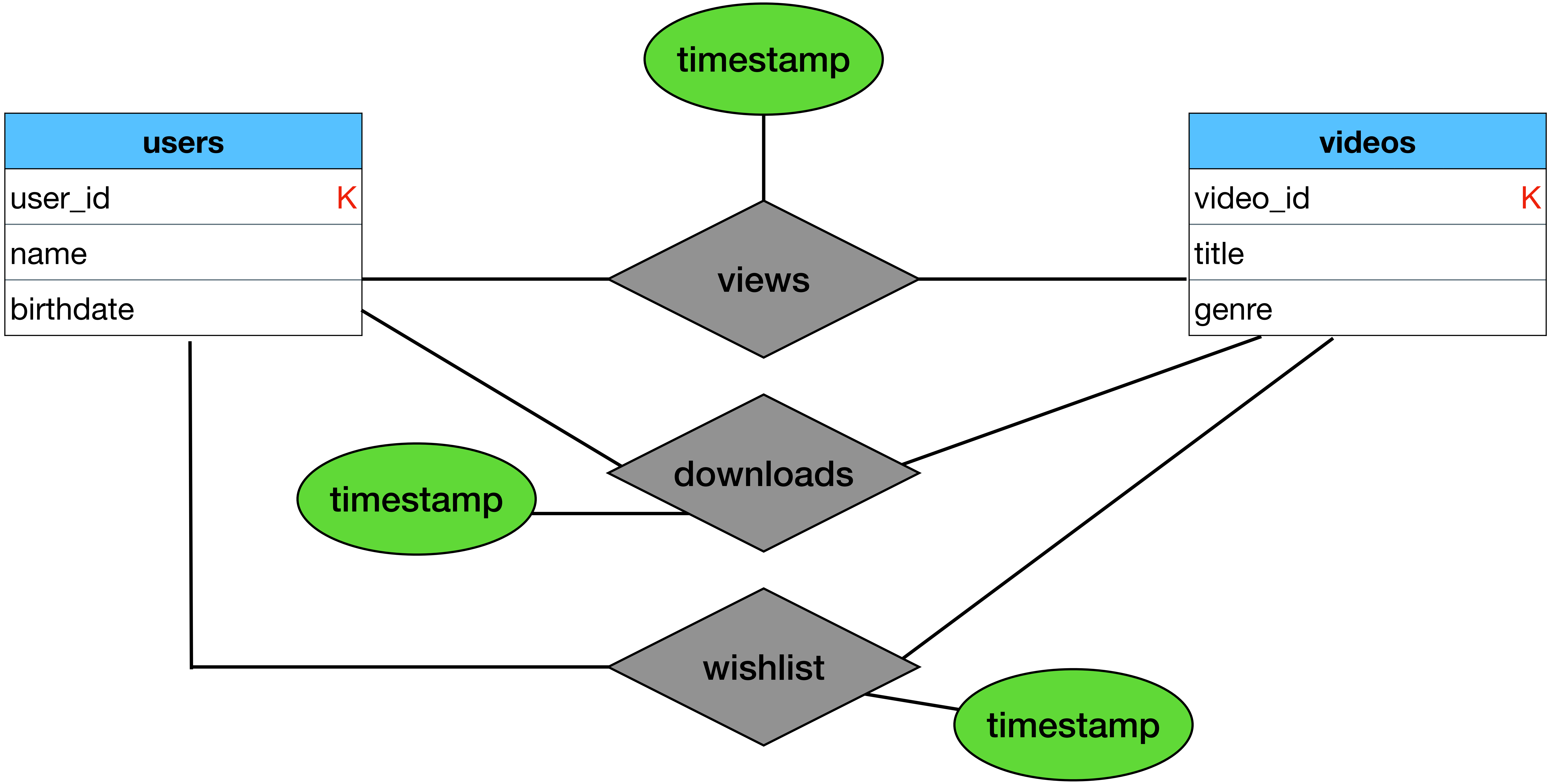
Example (6)

- How would the tables look like for both versions?



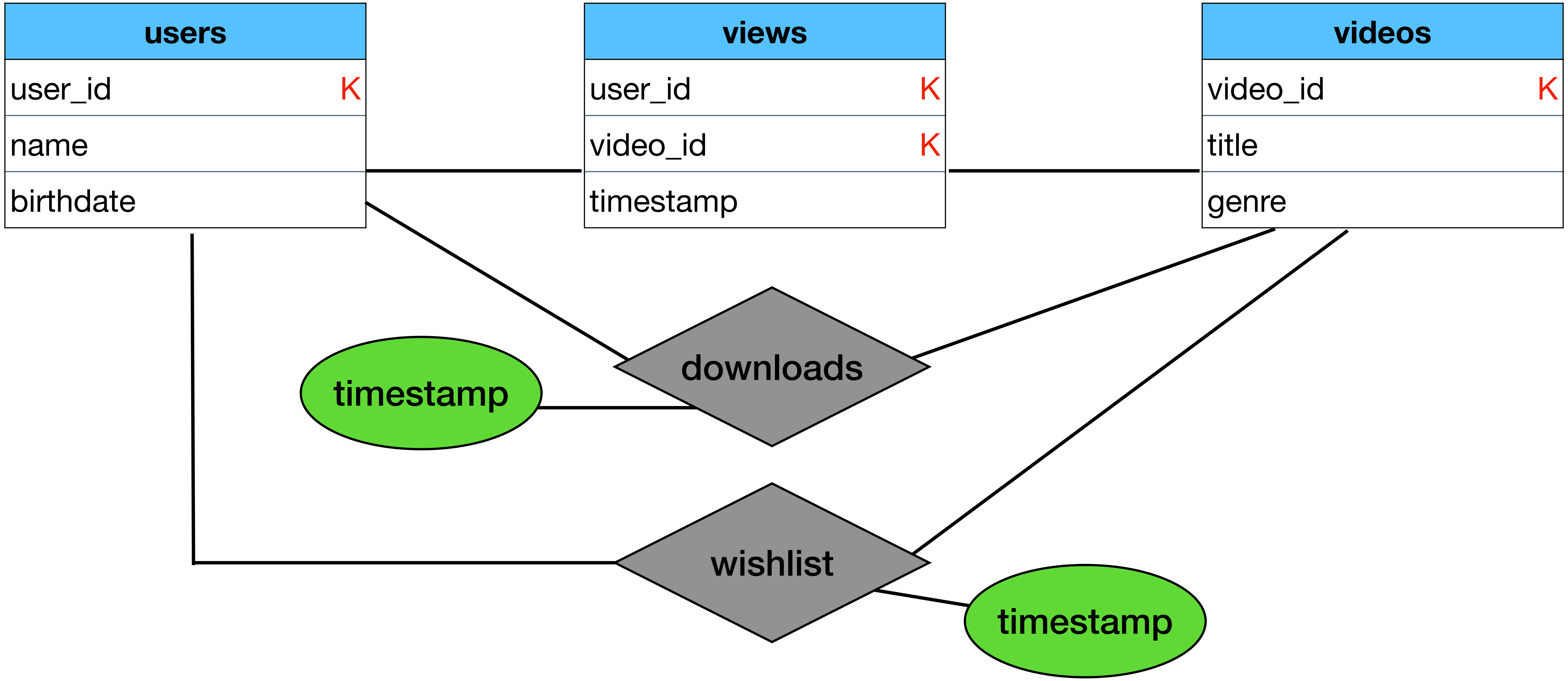
Example (6)

- How would the tables look like for both versions?



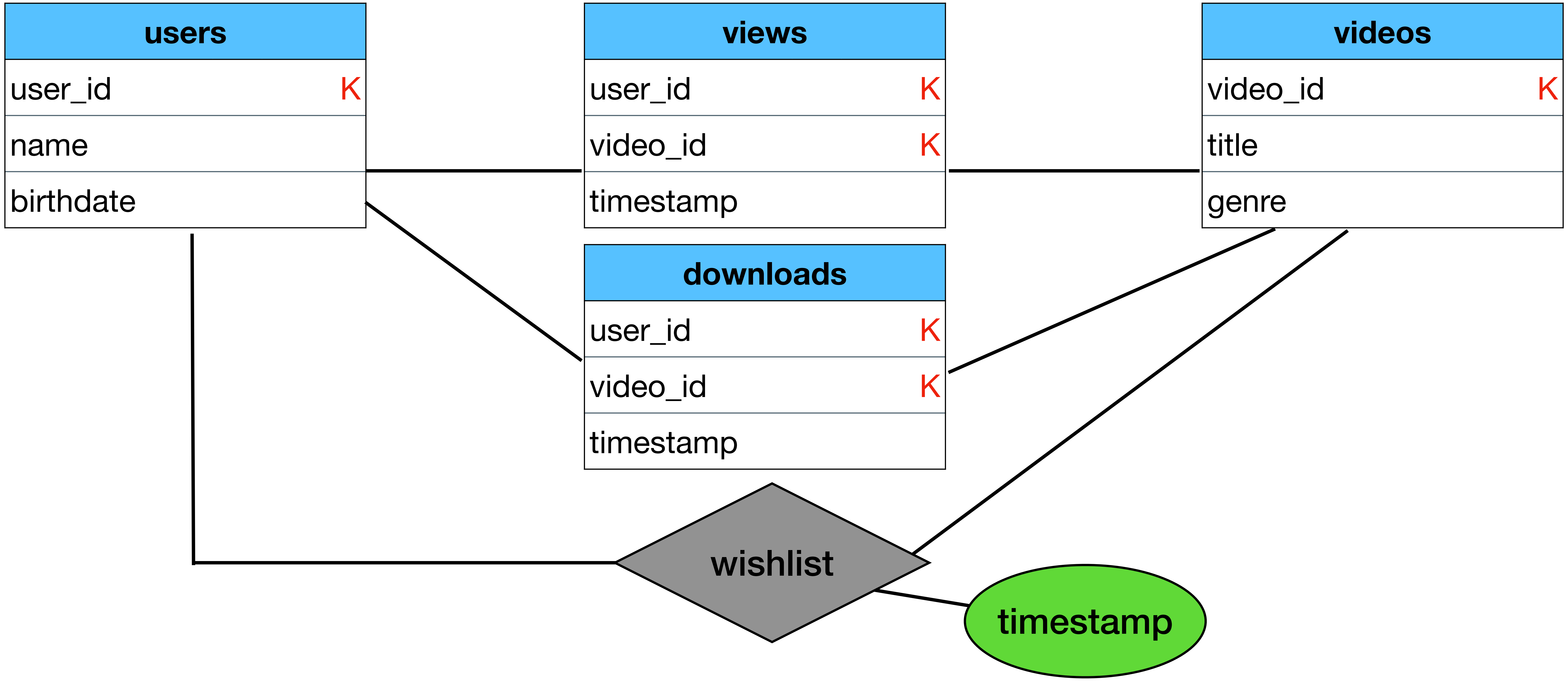
Example (6)

- How would the tables look like for both versions?



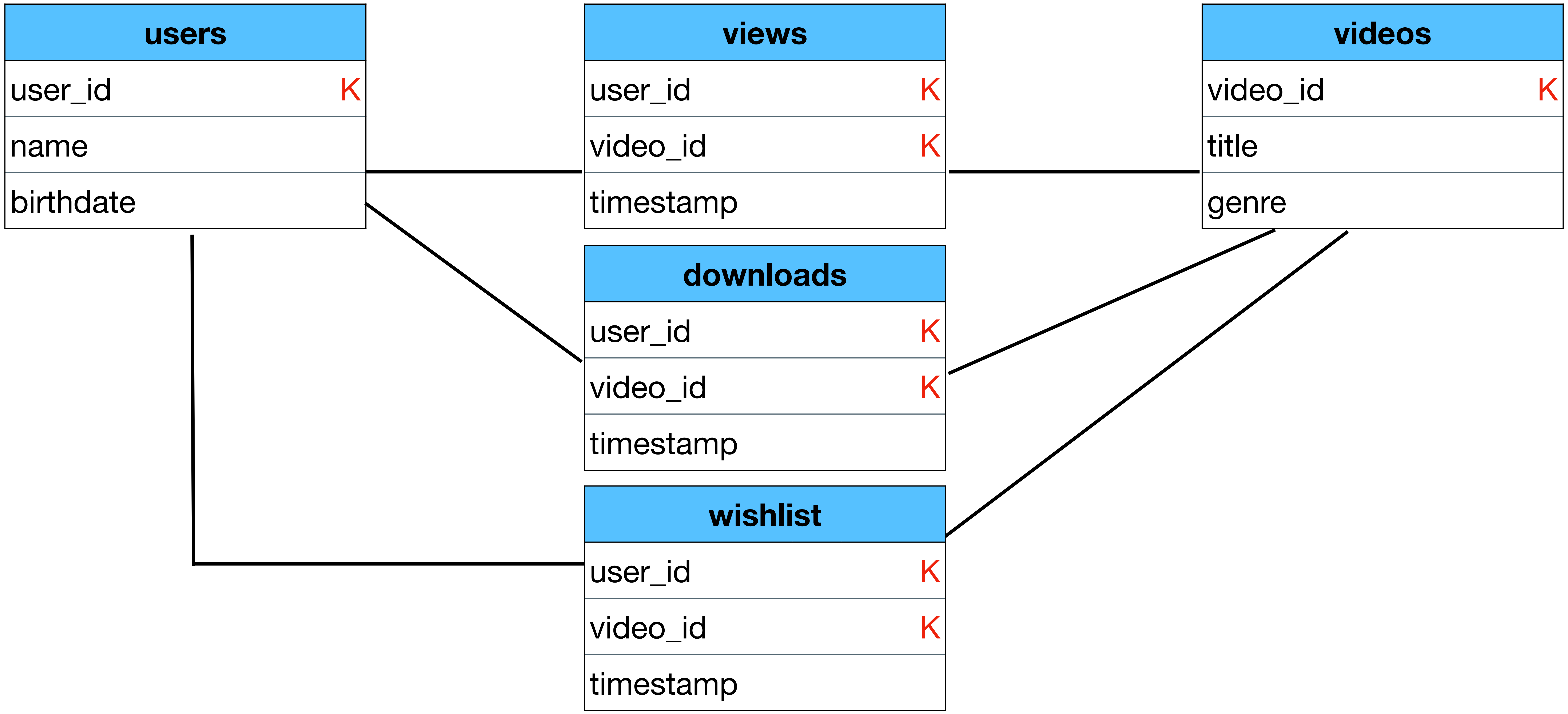
Example (6)

- How would the tables look like for both versions?



Example (6)

- How would the tables look like for both versions?



users	
user_id	K
name	
birthdate	

views	
user_id	K
video_id	K
timestamp	

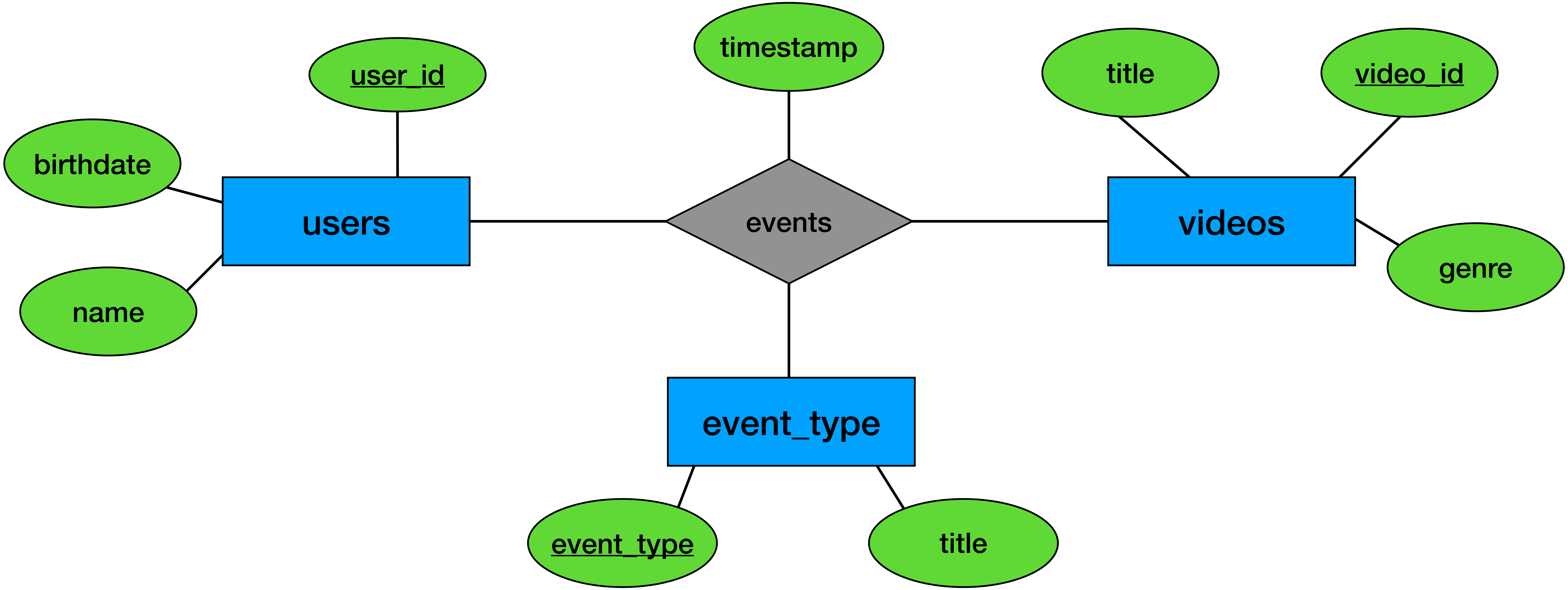
videos	
video_id	K
title	
genre	

downloads	
user_id	K
video_id	K
timestamp	

wishlist	
user_id	K
video_id	K
timestamp	

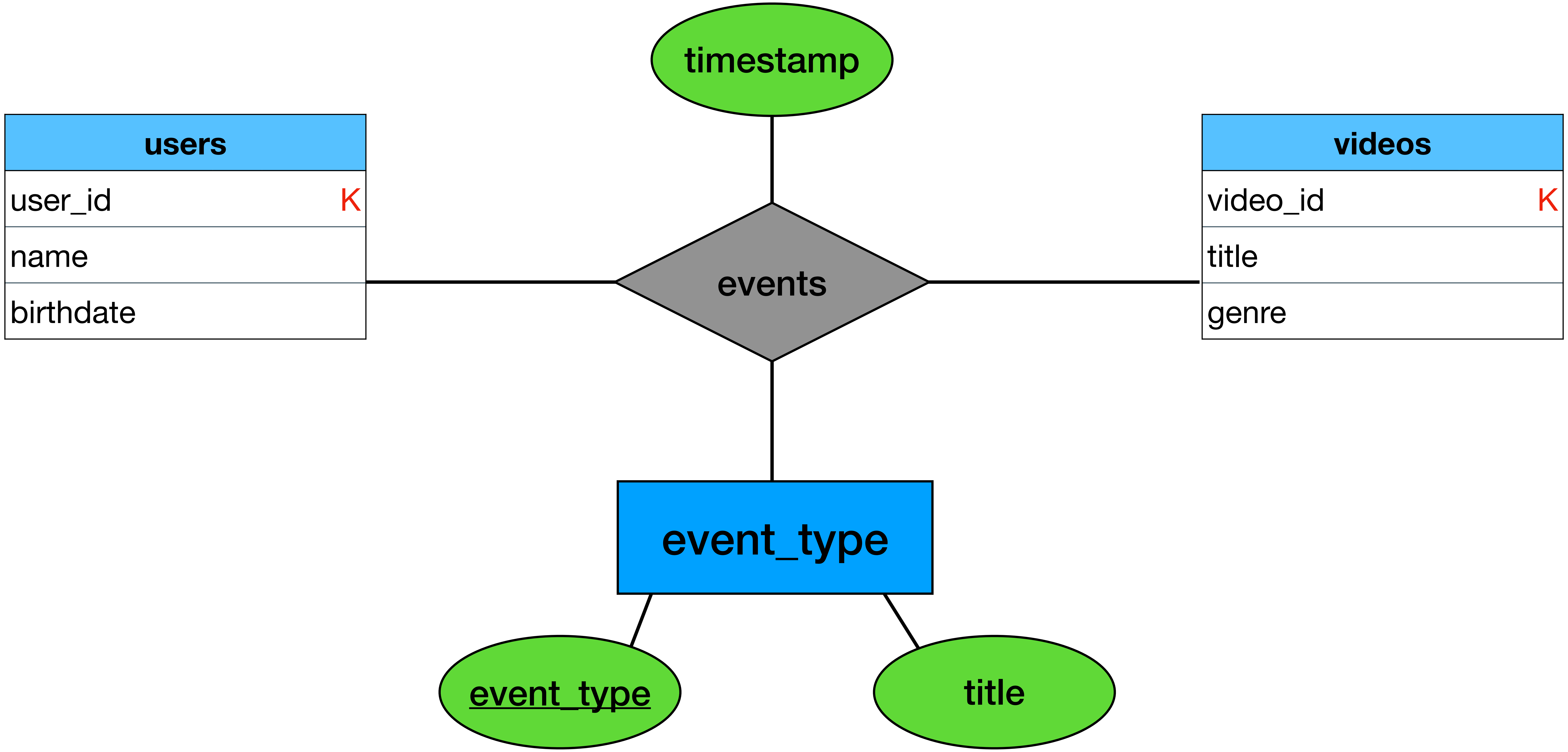
Example (6)

- How would the tables look like for both versions?



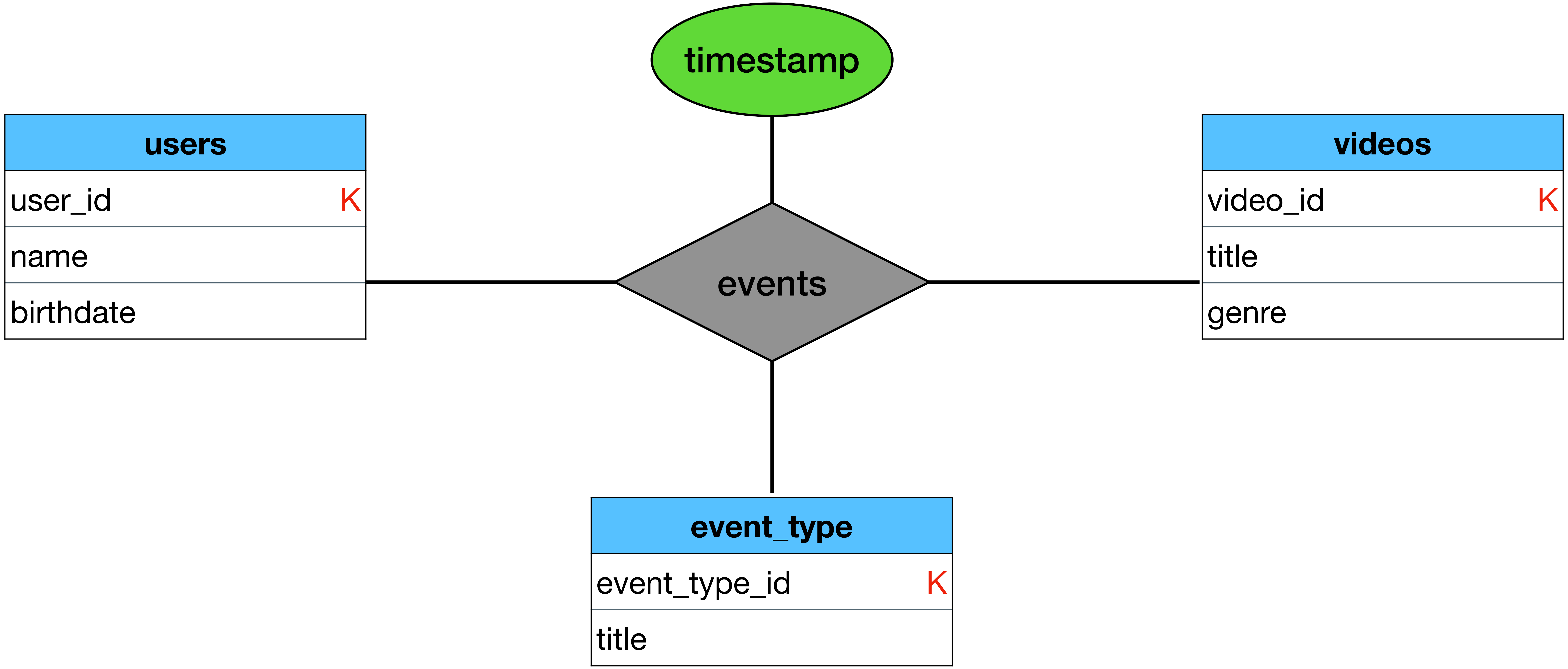
Example (6)

- How would the tables look like for both versions?



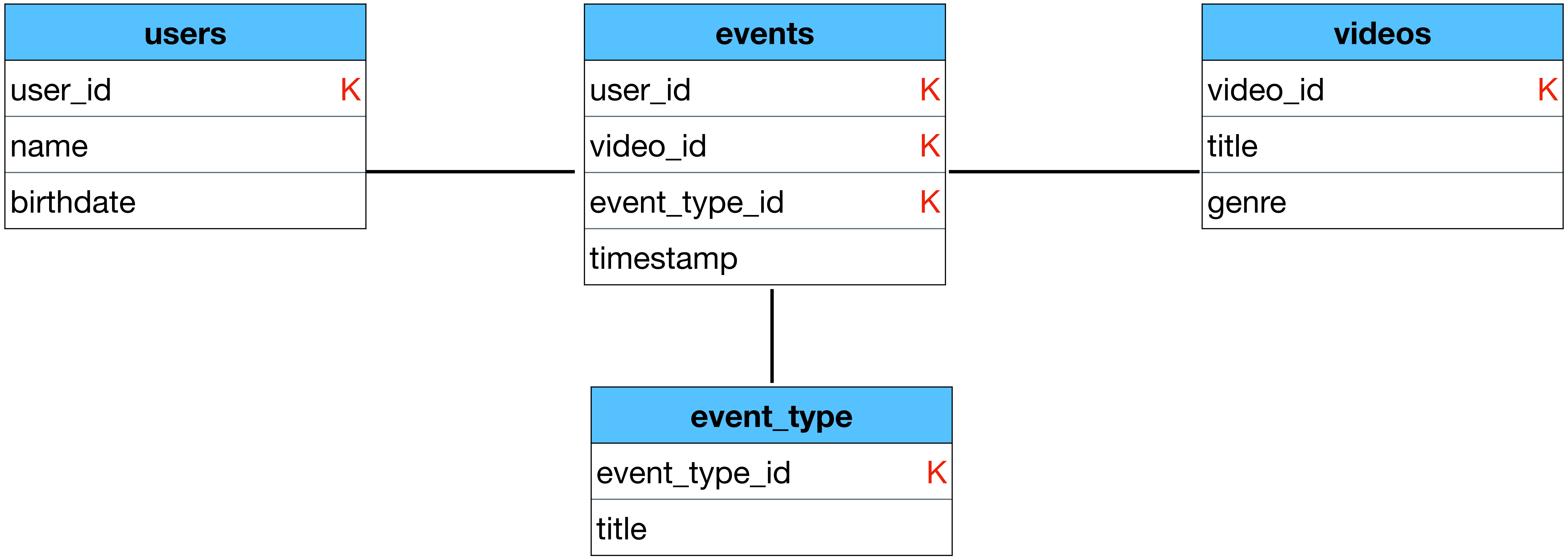
Example (6)

- How would the tables look like for both versions?



Example (6)

- How would the tables look like for both versions?



users	
user_id	K
name	
birthdate	

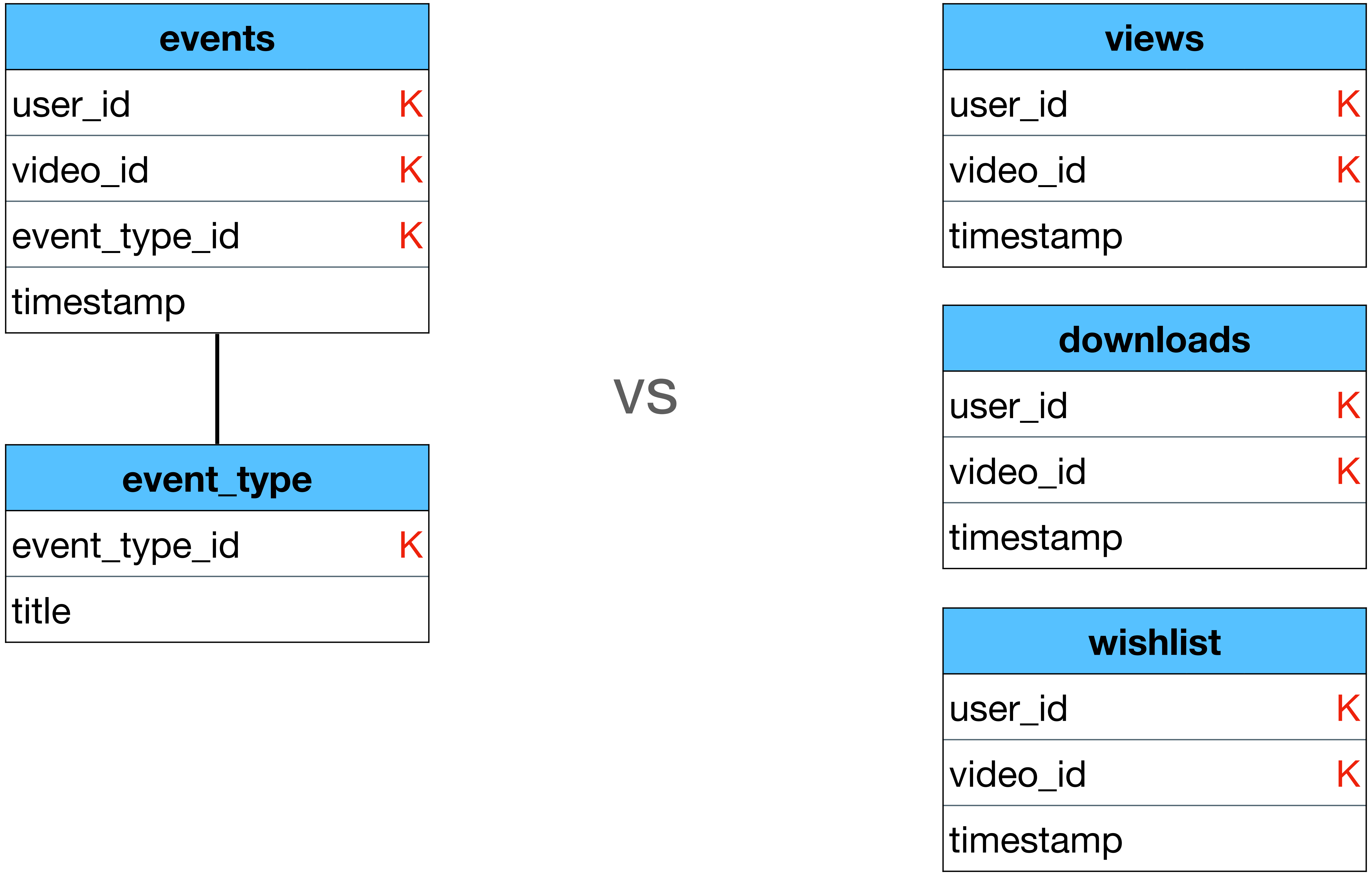
events	
user_id	K
video_id	K
event_type_id	K
timestamp	

videos	
video_id	K
title	
genre	

event_type	
event_type_id	K
title	

Example (6)

- So which version is better?



events	
user_id	K
video_id	K
event_type_id	K
timestamp	

event_type	
event_type_id	K
title	

vs

views	
user_id	K
video_id	K
timestamp	

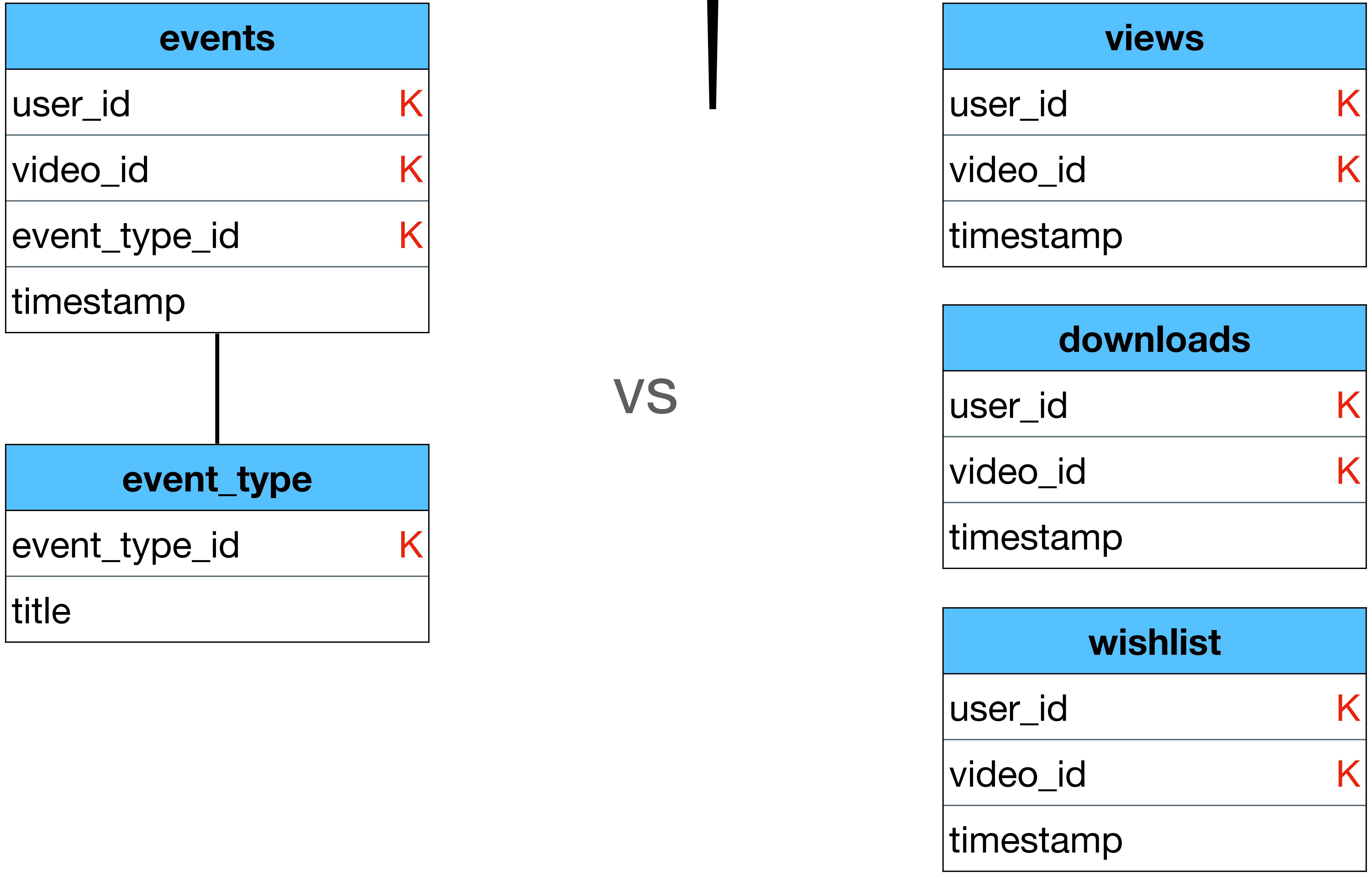
downloads	
user_id	K
video_id	K
timestamp	

wishlist	
user_id	K
video_id	K
timestamp	

Example (6)

If we might have new types of events in the future

- So which version is better?



events	
user_id	K
video_id	K
event_type_id	K
timestamp	

event_type	
event_type_id	K
title	

vs

views	
user_id	K
video_id	K
timestamp	

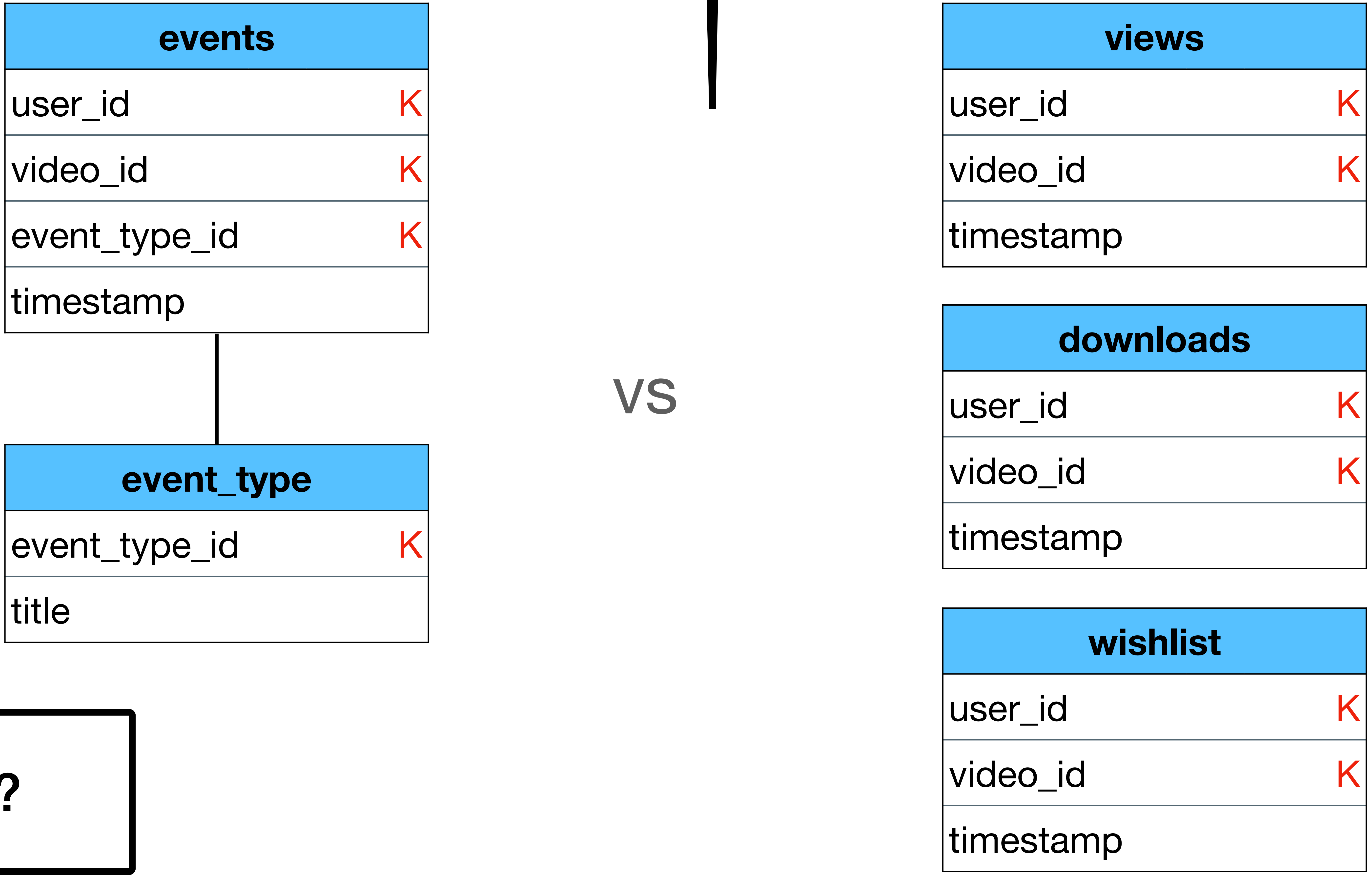
downloads	
user_id	K
video_id	K
timestamp	

wishlist	
user_id	K
video_id	K
timestamp	

Example (6)

- So which version is better?

If we might have new types of events in the future



events	
user_id	K
video_id	K
event_type_id	K
timestamp	

event_type	
event_type_id	K
title	

vs

views	
user_id	K
video_id	K
timestamp	

downloads	
user_id	K
video_id	K
timestamp	

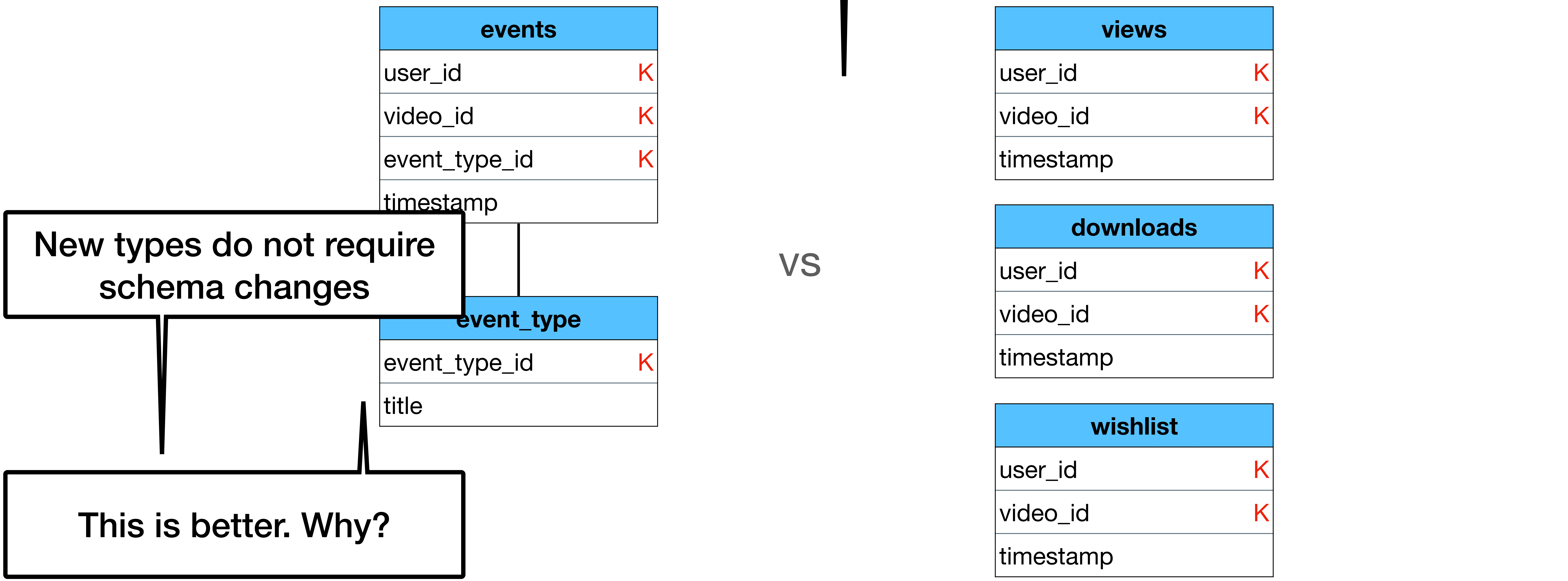
wishlist	
user_id	K
video_id	K
timestamp	

This is better. Why?

Example (6)

- So which version is better?

If we might have new types of events in the future



events	
user_id	K
video_id	K
event_type_id	K
timestamp	

event_type	
event_type_id	K
title	

vs

views	
user_id	K
video_id	K
timestamp	

downloads	
user_id	K
video_id	K
timestamp	

wishlist	
user_id	K
video_id	K
timestamp	

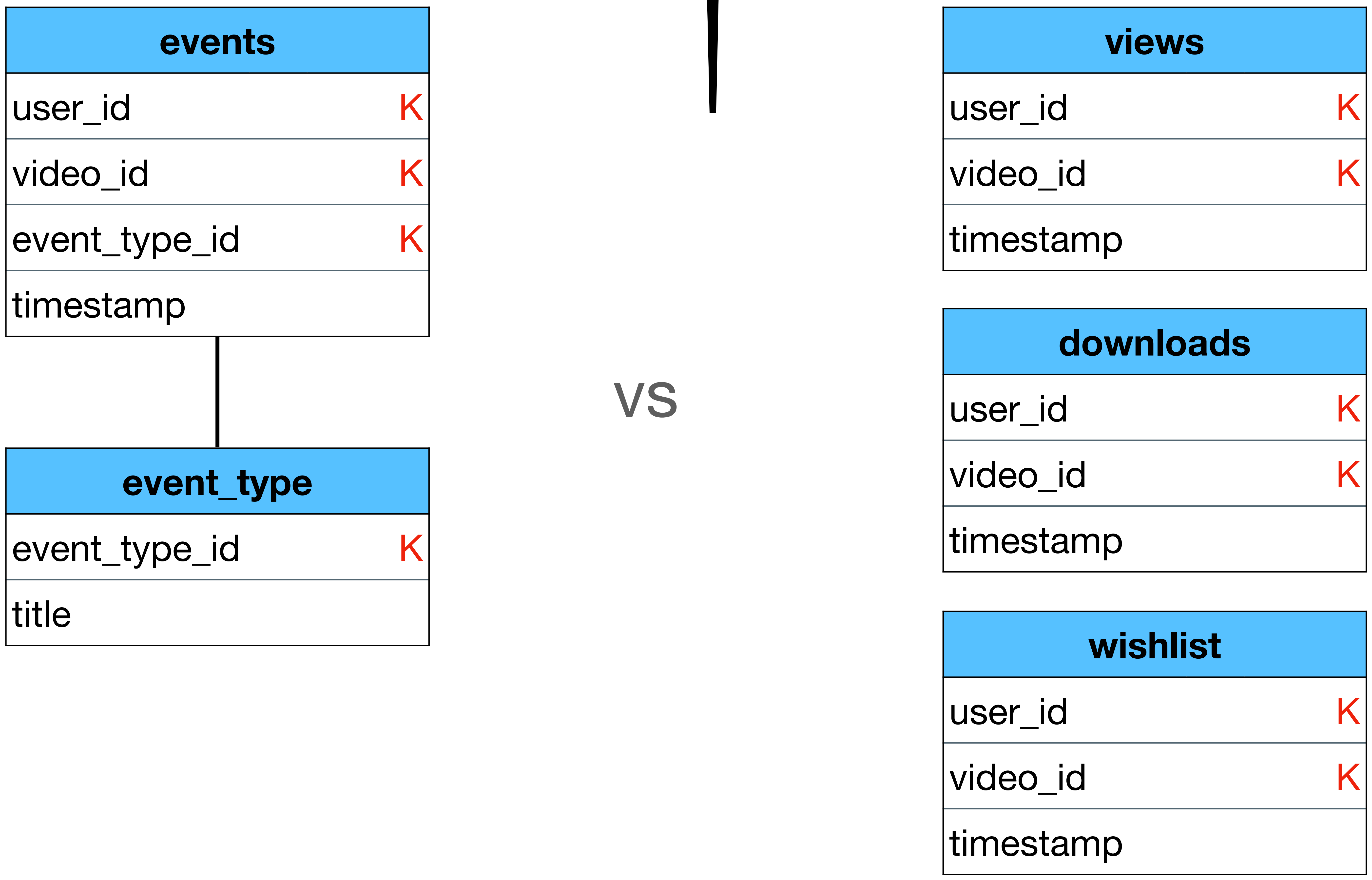
New types do not require schema changes

This is better. Why?

Example (6)

- So which version is better?

Not all dev teams have access to “views” data



events	
user_id	K
video_id	K
event_type_id	K
timestamp	

event_type	
event_type_id	K
title	

vs

views	
user_id	K
video_id	K
timestamp	

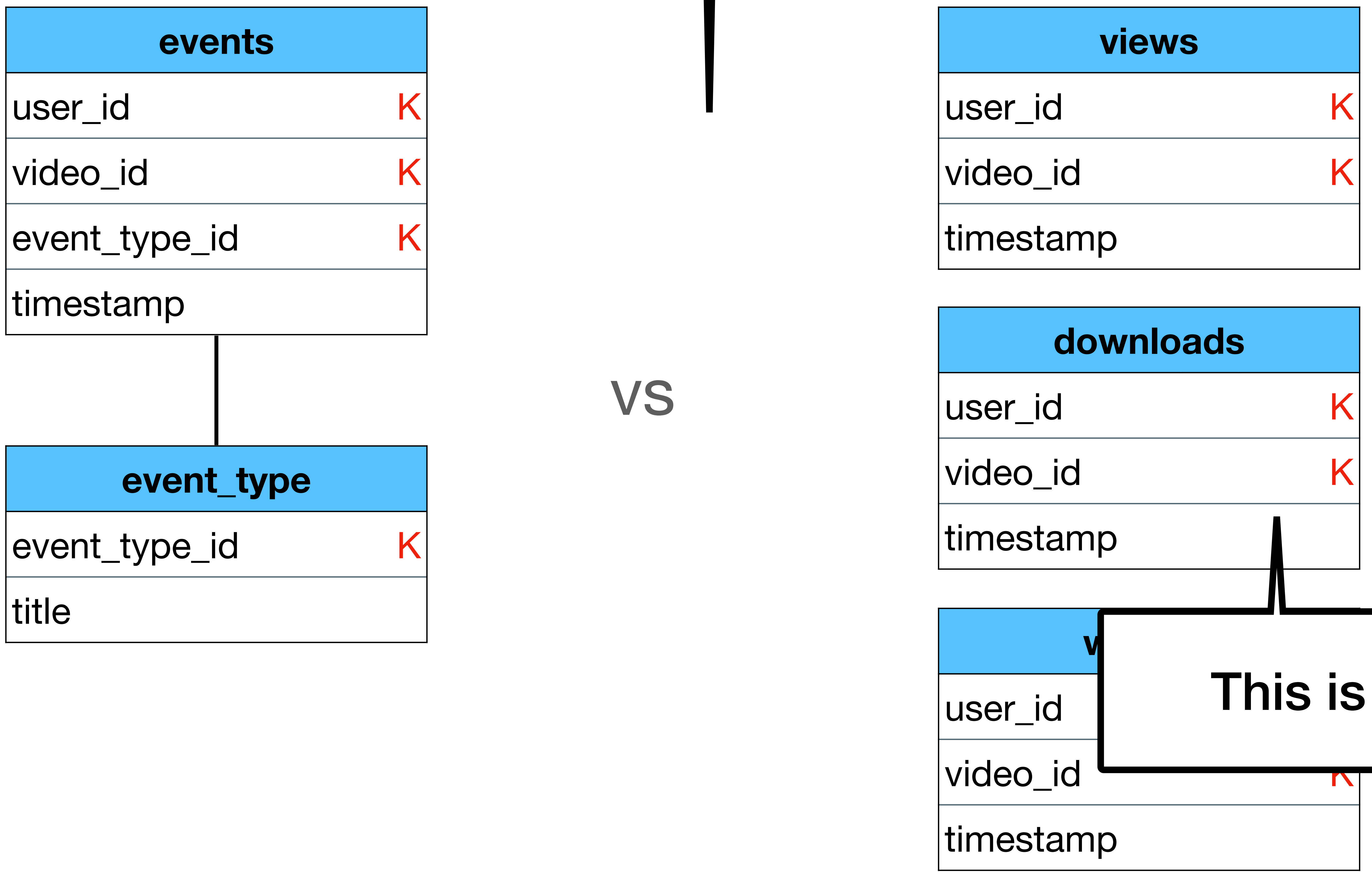
downloads	
user_id	K
video_id	K
timestamp	

wishlist	
user_id	K
video_id	K
timestamp	

Example (6)

- So which version is better?

Not all dev teams have access to “views” data



events	
user_id	K
video_id	K
event_type_id	K
timestamp	

event_type	
event_type_id	K
title	

vs

views	
user_id	K
video_id	K
timestamp	

downloads	
user_id	K
video_id	K
timestamp	

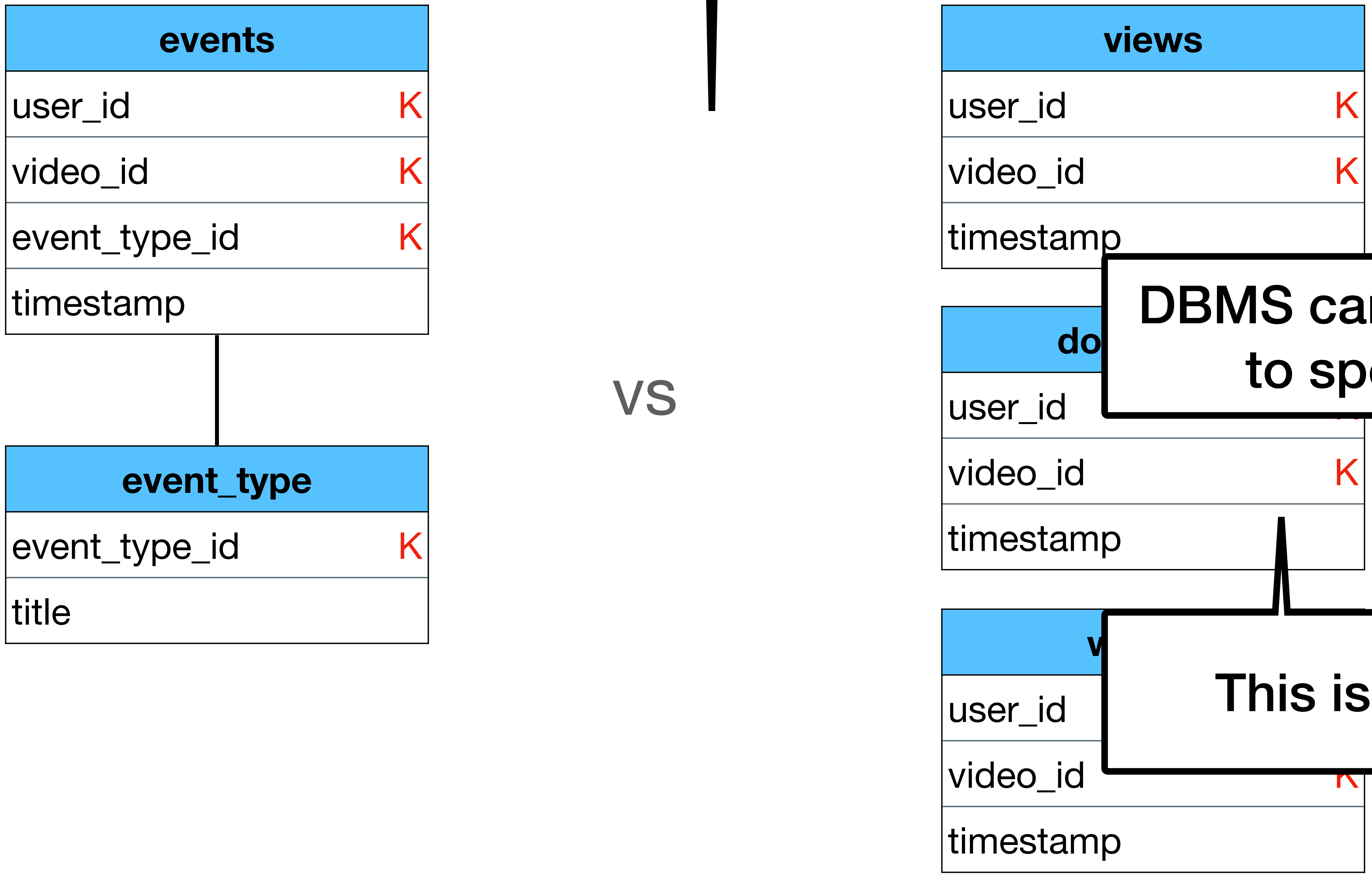
v	
user_id	
video_id	K
timestamp	

This is better. Why?

Example (6)

- So which version is better?

Not all dev teams have access to “views” data



events	
user_id	K
video_id	K
event_type_id	K
timestamp	

event_type	
event_type_id	K
title	

vs

views	
user_id	K
video_id	K
timestamp	

do	
user_id	
video_id	K
timestamp	

v	
user_id	
video_id	
timestamp	

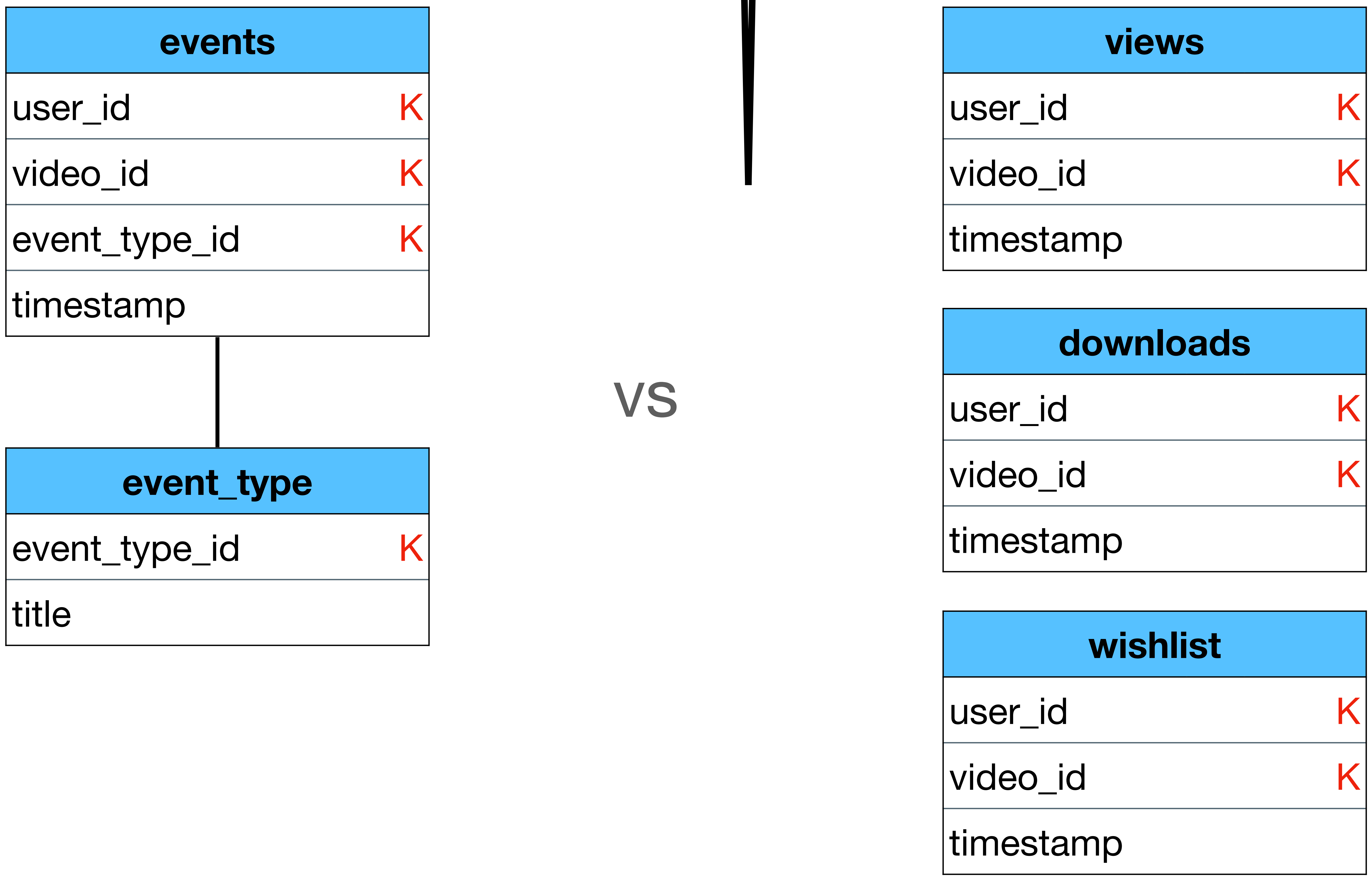
DBMS can restrict access to specific tables

This is better. Why?

Example (6)

- So which version is better?

Assume most of our queries requires only the wishlist data.
How many queries we need for each version?
How much each query “cost”?



events	
user_id	K
video_id	K
event_type_id	K
timestamp	

event_type	
event_type_id	K
title	

vs

views	
user_id	K
video_id	K
timestamp	

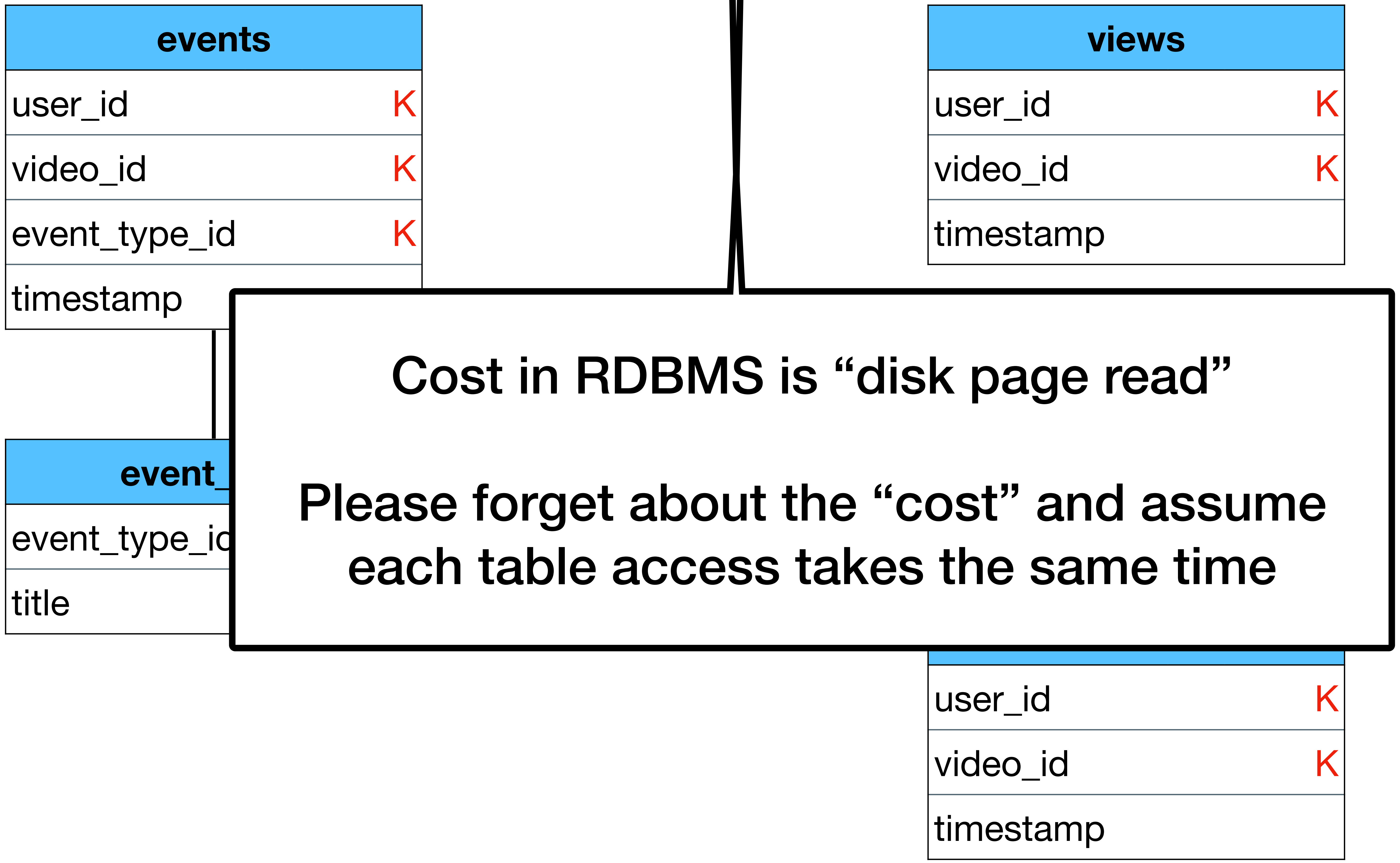
downloads	
user_id	K
video_id	K
timestamp	

wishlist	
user_id	K
video_id	K
timestamp	

Example (6)

- So which version is better?

Assume most of our queries requires only the wishlist data.
How many queries we need for each version?
How much each query “cost”?



events	
user_id	K
video_id	K
event_type_id	K
timestamp	

event_
event_type_ic
title

views	
user_id	K
video_id	K
timestamp	

user_id	K
video_id	K
timestamp	

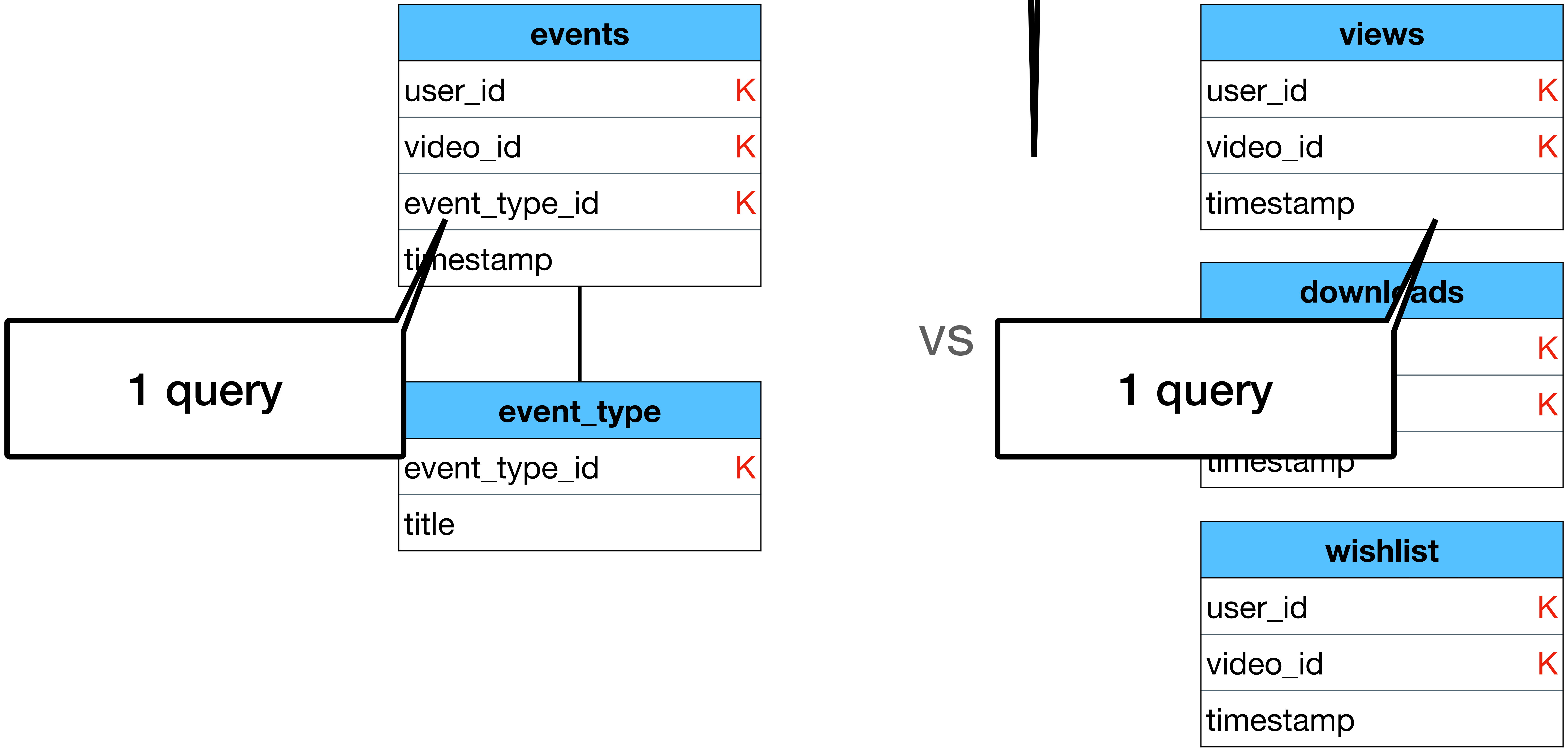
Cost in RDBMS is “disk page read”

Please forget about the “cost” and assume each table access takes the same time

Example (6)

- So which version is better?

Assume most of our queries requires only the wishlist data.
How many queries we need for each version?
How much each query "cost"?



events	
user_id	K
video_id	K
event_type_id	K
timestamp	

event_type	
event_type_id	K
title	

1 query

vs

views	
user_id	K
video_id	K
timestamp	

downloads	
	K
	K
timestamp	

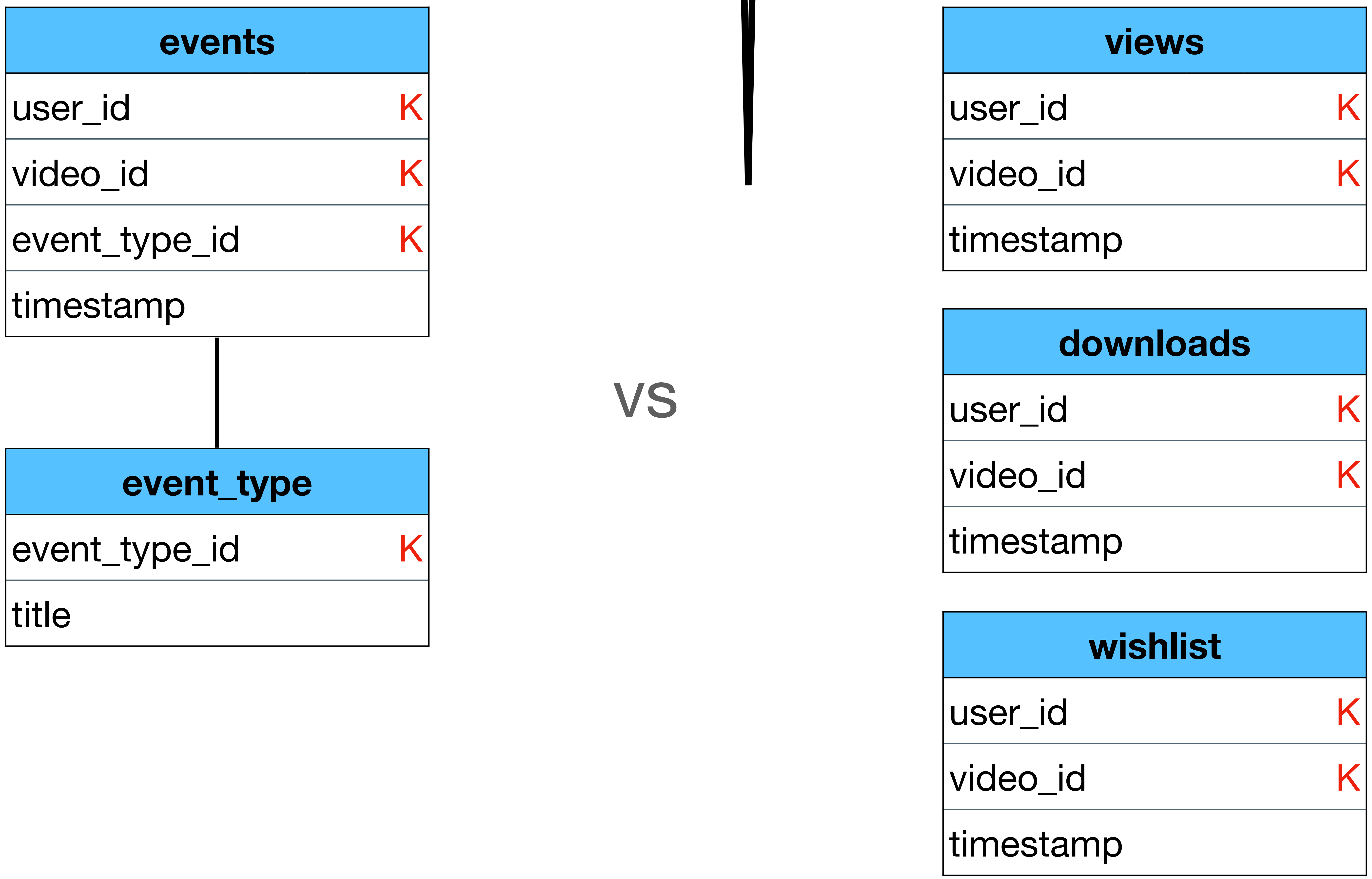
wishlist	
user_id	K
video_id	K
timestamp	

1 query

Example (6)

- So which version is better?

Assume most of our queries requires only the **wishlist** data AND the **downloads**
How many queries we need for each version?



events	
user_id	K
video_id	K
event_type_id	K
timestamp	

event_type	
event_type_id	K
title	

vs

views	
user_id	K
video_id	K
timestamp	

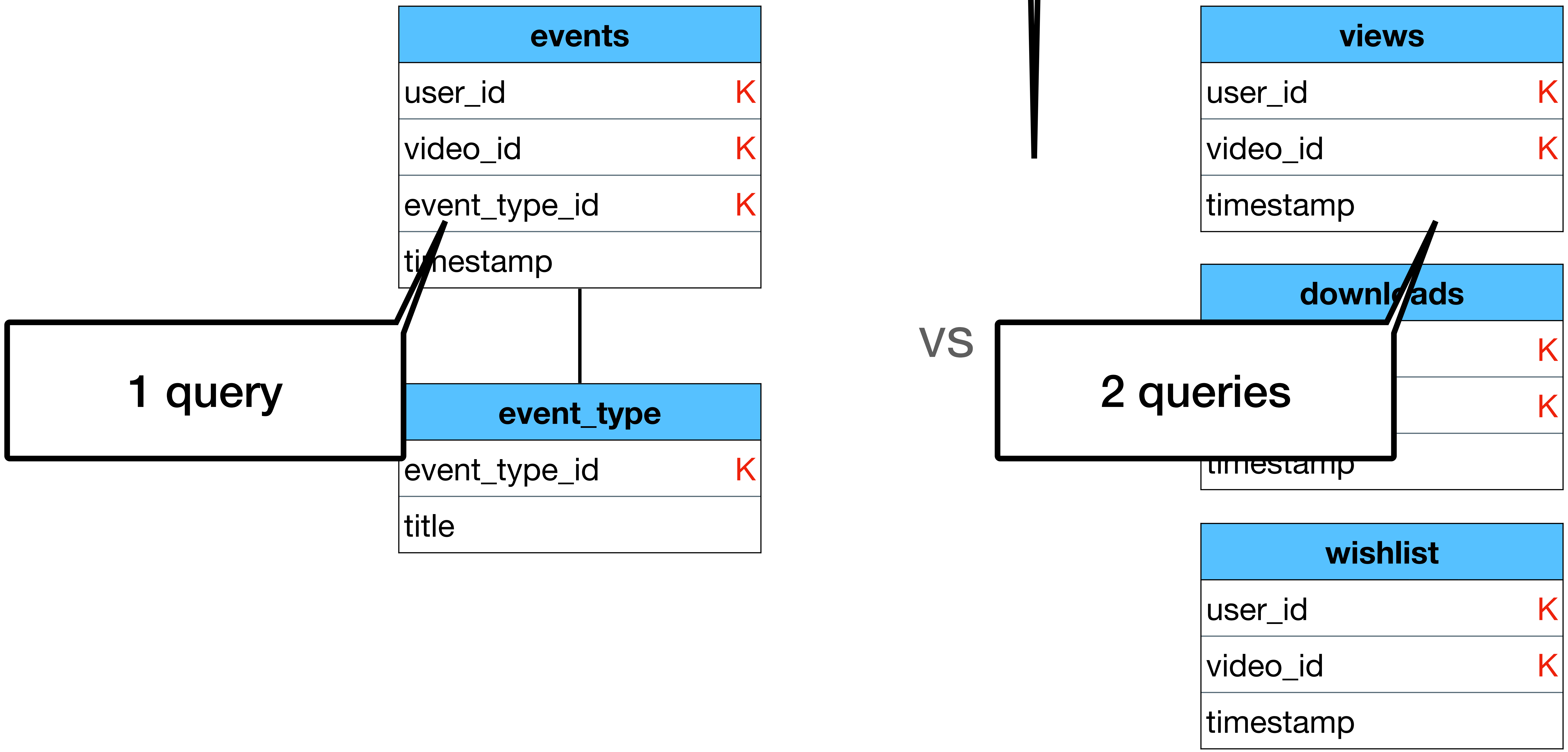
downloads	
user_id	K
video_id	K
timestamp	

wishlist	
user_id	K
video_id	K
timestamp	

Example (6)

- So which version is better?

Assume most of our queries requires only the **wishlist** data AND the **downloads**
How many queries we need for each version?



events	
user_id	K
video_id	K
event_type_id	K
timestamp	

event_type	
event_type_id	K
title	

1 query

vs

views	
user_id	K
video_id	K
timestamp	

downloads	
	K
	K
timestamp	

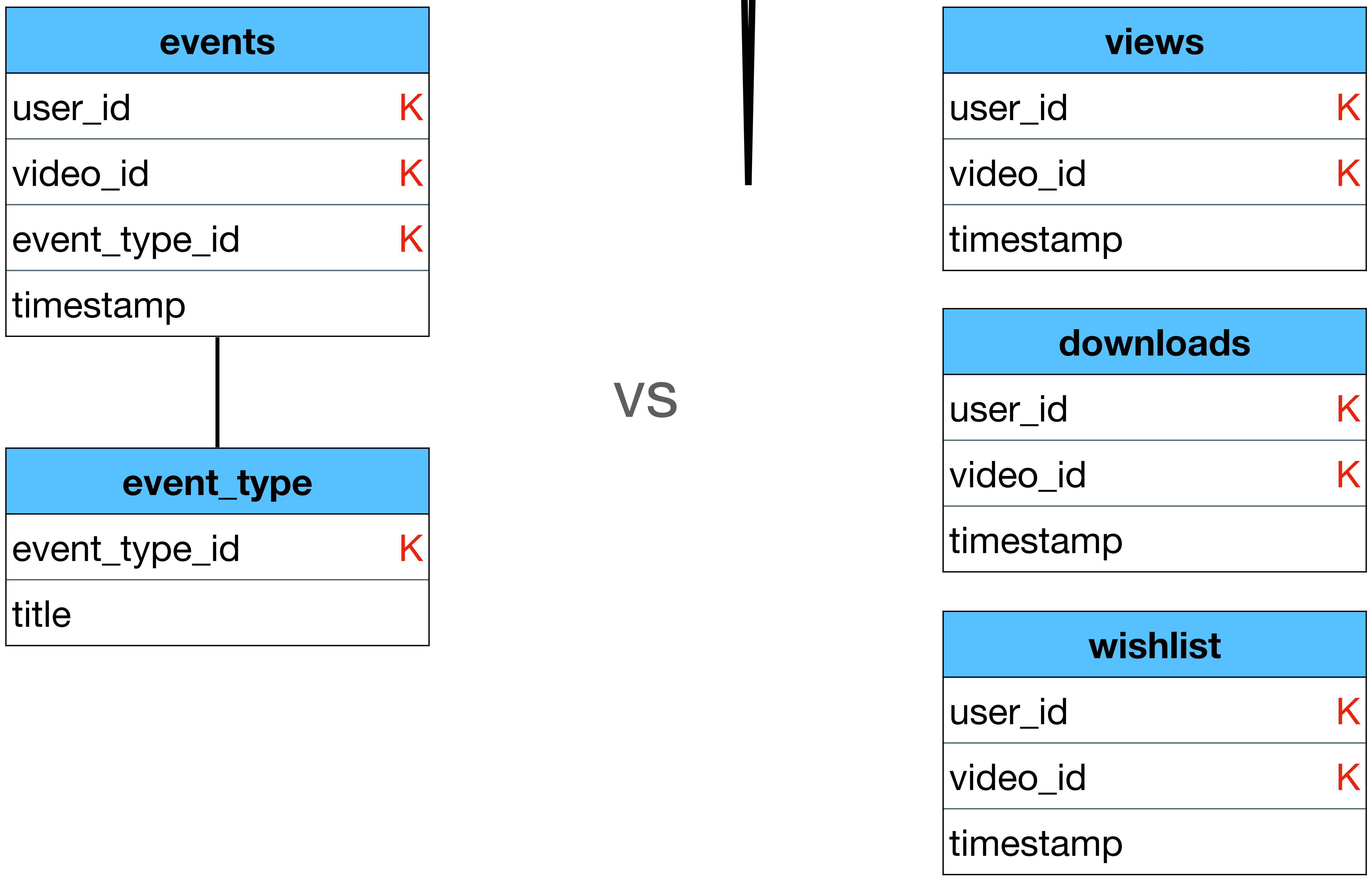
wishlist	
user_id	K
video_id	K
timestamp	

2 queries

Example (6)

Assume most of our queries requires only the **wishlist** data AND the **downloads** AND the **views**
How many queries we need for each version?

- So which version is better?



events	
user_id	K
video_id	K
event_type_id	K
timestamp	

event_type	
event_type_id	K
title	

VS

views	
user_id	K
video_id	K
timestamp	

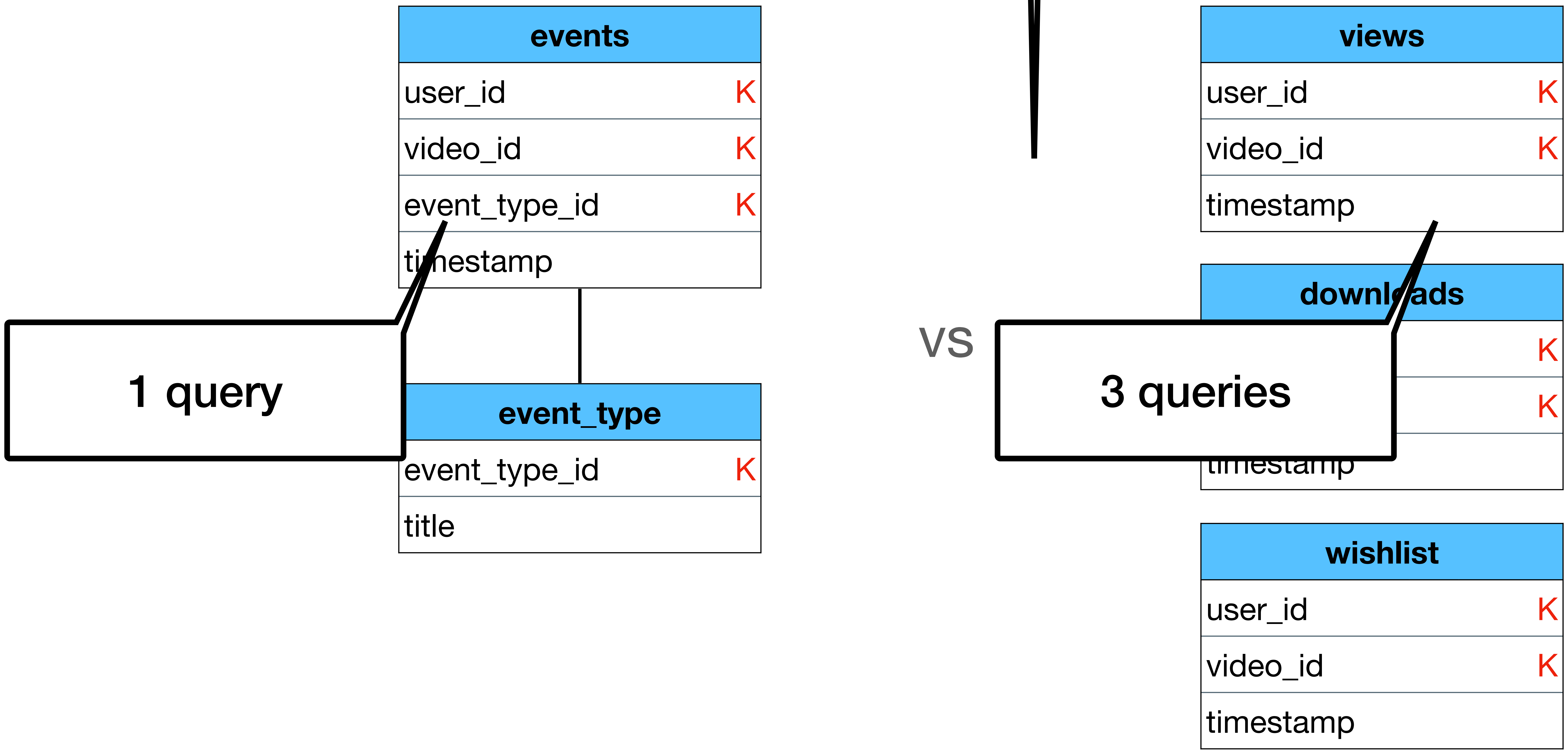
downloads	
user_id	K
video_id	K
timestamp	

wishlist	
user_id	K
video_id	K
timestamp	

Example (6)

- So which version is better?

Assume most of our queries requires only the **wishlist** data AND the **downloads** AND the **views**
How many queries we need for each version?



events

user_id	K
video_id	K
event_type_id	K
timestamp	

event_type

event_type_id	K
title	

1 query

vs

views

user_id	K
video_id	K
timestamp	

downloads

	K
	K
timestamp	

wishlist

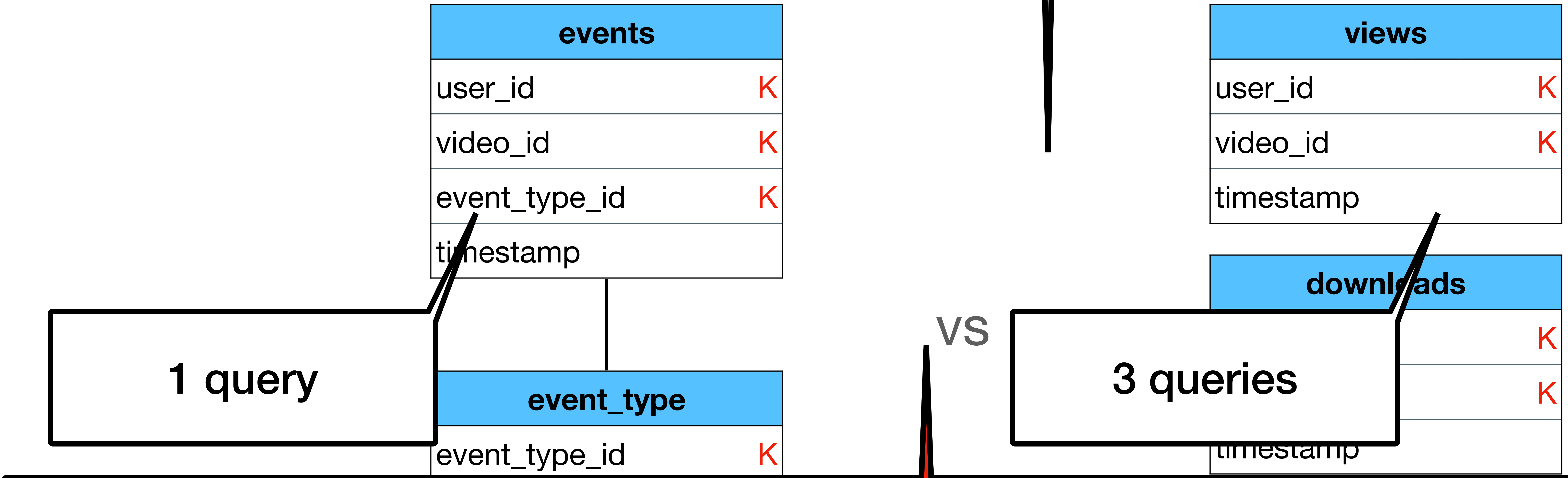
user_id	K
video_id	K
timestamp	

3 queries

Example (6)

Assume most of our queries requires only the **wishlist** data AND the **downloads** AND the **views**
How many queries we need for each version?

- So which version is better?



1 query

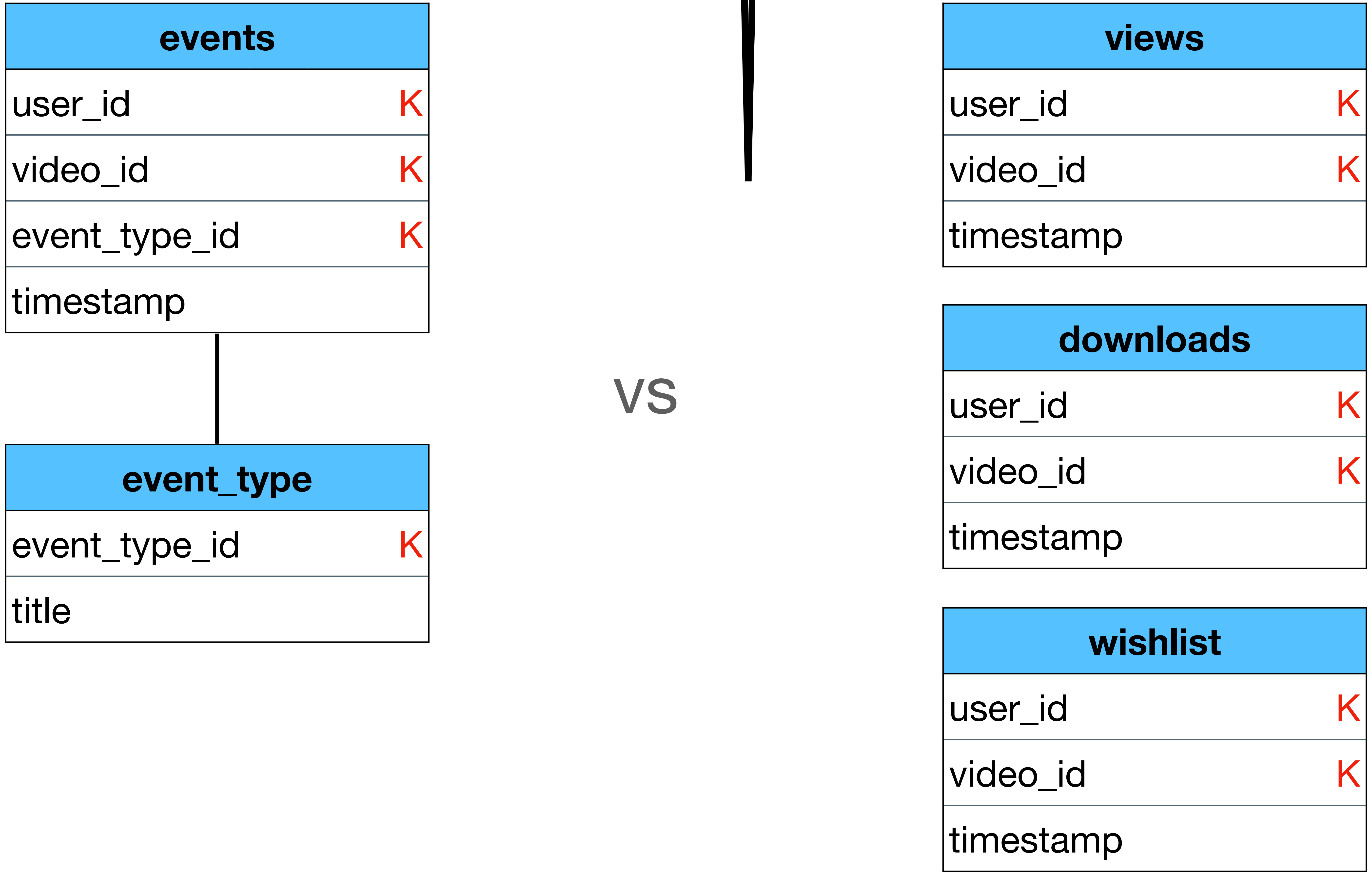
3 queries

This is actually not true - it depends on how the data is stored on disk.
<u>We will talk about this over and over in the next lessons :)</u>

Example (6)

- So which version is better?

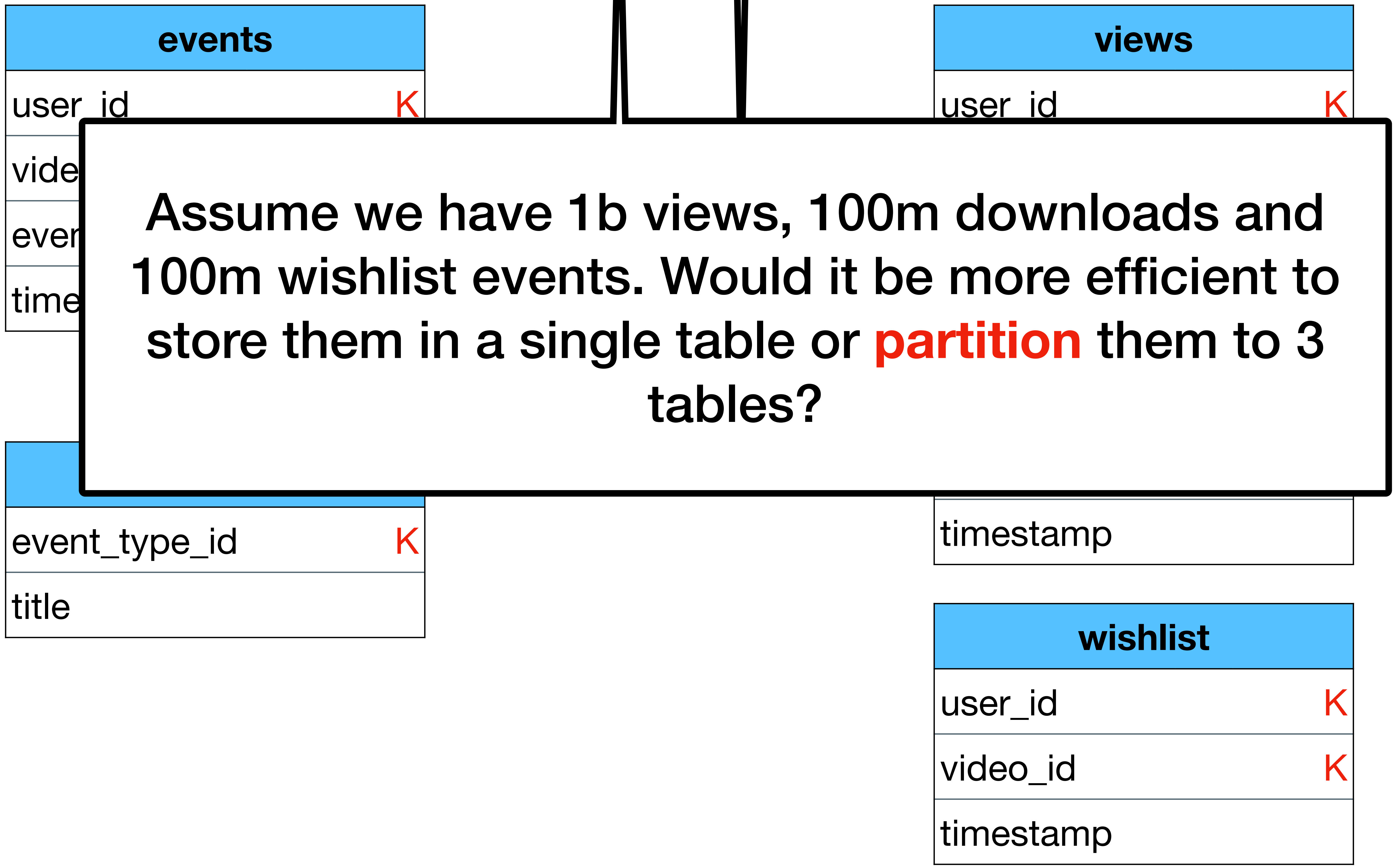
Assume events have different distributions.
For each 10 views there is 1 download and 1 wishlist events
Would you change your previous answers?



Example (6)

- So which version is better?

Assume events have different distributions.
For each 10 views there is 1 download and 1 wishlist events
Would you change your previous answers?



events	
user_id	K
vide	
ever	
time	

event_type_id	K
title	

views	
user_id	K
timestamp	

wishlist	
user_id	K
video_id	K
timestamp	

Assume we have 1b views, 100m downloads and 100m wishlist events. Would it be more efficient to store them in a single table or partition them to 3 tables?

Example (6)

- So which version is better?

Assume events have different distributions.
For each 10 views there is 1 download and 1 wishlist events
Would you change your previous answers?

events	
user_id	K
vide	
ever	
time	

event_type_id	K

views	
user_id	K
timestamp	

wishlist	
user_id	K
video_id	K
timestamp	

Assume we have 1b views, 100m downloads and 100m wishlist events. Would it be more efficient to store them in a single table or partition them to 3 tables?

Doesn't really matter because a table with 1b rows will probably "break" the RDBMS
(Unless you are Facebook or Amazon)

Example (6)

Assume events have different distributions.
For each 10 views there is 1 download and 1 wishlist events
Would you change your previous answers?

- So which version is better?

events

views

Don’t worry - this is the “Big Data System” course, not “Database Systems”.
We will solve this soon :)

store them in a single table or partition them to 3 tables?

event_type_id	K

timestamp

wishlist	
user_id	K
video_id	K
timestamp	

Doesn’t really matter because a table with 1b rows will probably “break” the RDBMS
(Unless you are Facebook or Amazon)