Denormalization

Big Data Systems

Dr. Rubi Boim

Motivation (for this course)

- The basics of wide column data modeling

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Denormalization

From Wikipedia, the free encyclopedia

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.

Find sources: "Denormalization" – news · newspapers · books · scholar · JSTOR *(May 2008)*

(Learn how and when to remove this template message)

Denormalization is a strategy used on a previously-normalized database to increase performance. In computing, denormalization is the process of trying to improve the read performance of a database, at the expense of losing some write performance, by adding redundant copies of data or by grouping data.[1][2] It is often motivated by performance or scalability in relational database software needing to carry out very large numbers of read operations. Denormalization differs from the unnormalized form in that denormalization benefits can only be fully realized on a data model that is otherwise normalized.

WIKIPEDIA
The Free Encyclopedia

Not logged in Talk Contributions Create account Log in

Article Talk | Read Edit View history | Search Wikipedia

Denormalization

From Wikipedia, the free encyclopedia

Also applied to Relational Databases

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.

Find sources: "Denormalization" – news · newspapers · books · scholar · JSTOR *(May 2008)*

(Learn how and when to remove this template message)

Denormalization is a strategy used on a previously-normalized database to increase performance. In computing, denormalization is the process of trying to improve the read performance of a database, at the expense of losing some write performance, by adding redundant copies of data or by grouping data.[1][2] It is often motivated by performance or scalability in relational database software needing to carry out very large numbers of read operations. Denormalization differs from the unnormalized form in that denormalization benefits can only be fully realized on a data model that is otherwise normalized.

What is normalized?

WIKIPEDIA
The Free Encyclopedia

Not logged in Talk Contributions Create account Log in

Article Talk | Read Edit View history | Search Wikipedia

Denormalization

From Wikipedia, the free encyclopedia

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.

Find sources: "Denormalization" – news · newspapers · books · scholar · JSTOR *(May 2008)*

(Learn how and when to remove this template message)

Denormalization is a strategy used on a previously-normalized database to increase performance. In computing, denormalization is the process of trying to improve the read performance of a database, at the expense of losing some write performance, by adding redundant copies of data or by grouping data.[1][2] It is often motivated by performance or scalability in relational database software needing to carry out very large numbers of read operations. Denormalization differs from the unnormalized form in that denormalization benefits can only be fully realized on a data model that is otherwise normalized.

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Functional dependency

- A constraint between two sets of attributes
  X—>Y

For example

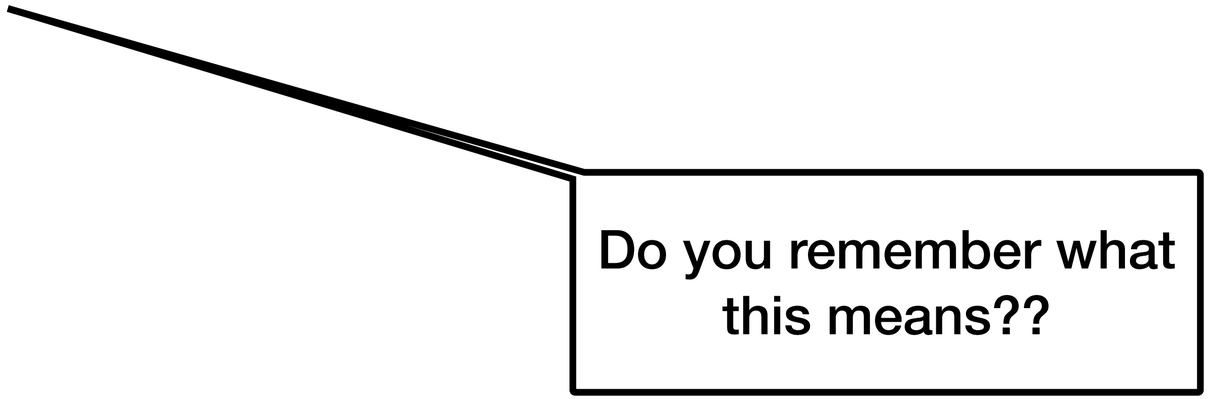
- {country} —> {continent}
  if we know the country is France, we know the continent is Europe
- {user_id} —> {user_name}
  if we know the user id is 123 we know it is “Rubi"

Relational DB & Normalization

- “requires” the data to be normalized in order to
  - Reduce data redundancy
  - Improve data integrity

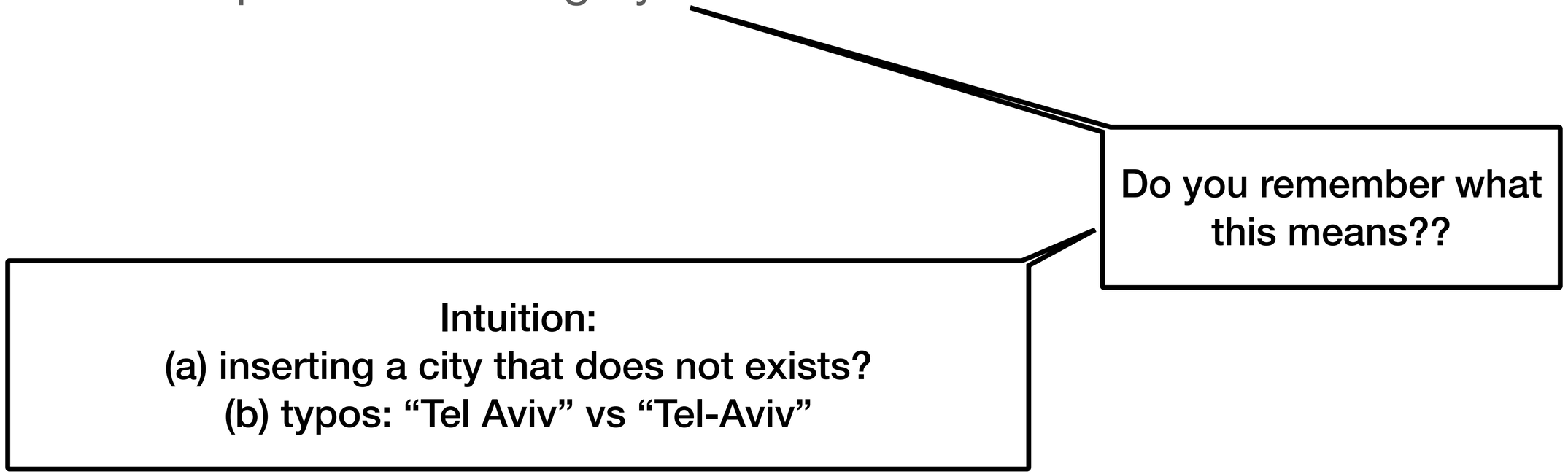
Relational DB & Normalization

- “requires” the data to be normalized in order to
  - Reduce data redundancy
  - Improve data integrity



Relational DB & Normalization

- “requires” the data to be normalized in order to
  - Reduce data redundancy
  - Improve data integrity



Relational DB & Normalization

- “requires” the data to be normalized in order to
  - Reduce data redundancy
  - Improve data integrity

A relation R is in 3rd normal form if:

Whenever there is a nontrivial dependency A1,A2,…,An —>B for R, then {A1, A2, …, An} a super key for R,
or B is part of a key

(See “Database Systems” course for more info)

Normalization - Example

users-ver1

user_id	fname	lname	city	country
101	Rubi	Boim	Tel Aviv	Israel
102	Tova	Milo	Tel Aviv	Israel
103	Lebron	James	Los Angeles	USA

What are the functional dependencies?

Normalization - Example

users-ver1

user_id	fname	lname	city	country
101	Rubi	Boim	Tel Aviv	Israel
102	Tova	Milo	Tel Aviv	Israel
103	Lebron	James	Los Angeles	USA

What are the functional dependencies?

{user_id} —> {fname,lname,city,country}
user_id is a key 👍🏻

Normalization - Example

users-ver1

user_id	fname	lname	city	country
101	Rubi	Boim	Tel Aviv	Israel
102	Tova	Milo	Tel Aviv	Israel
103	Lebron	James	Los Angeles	USA

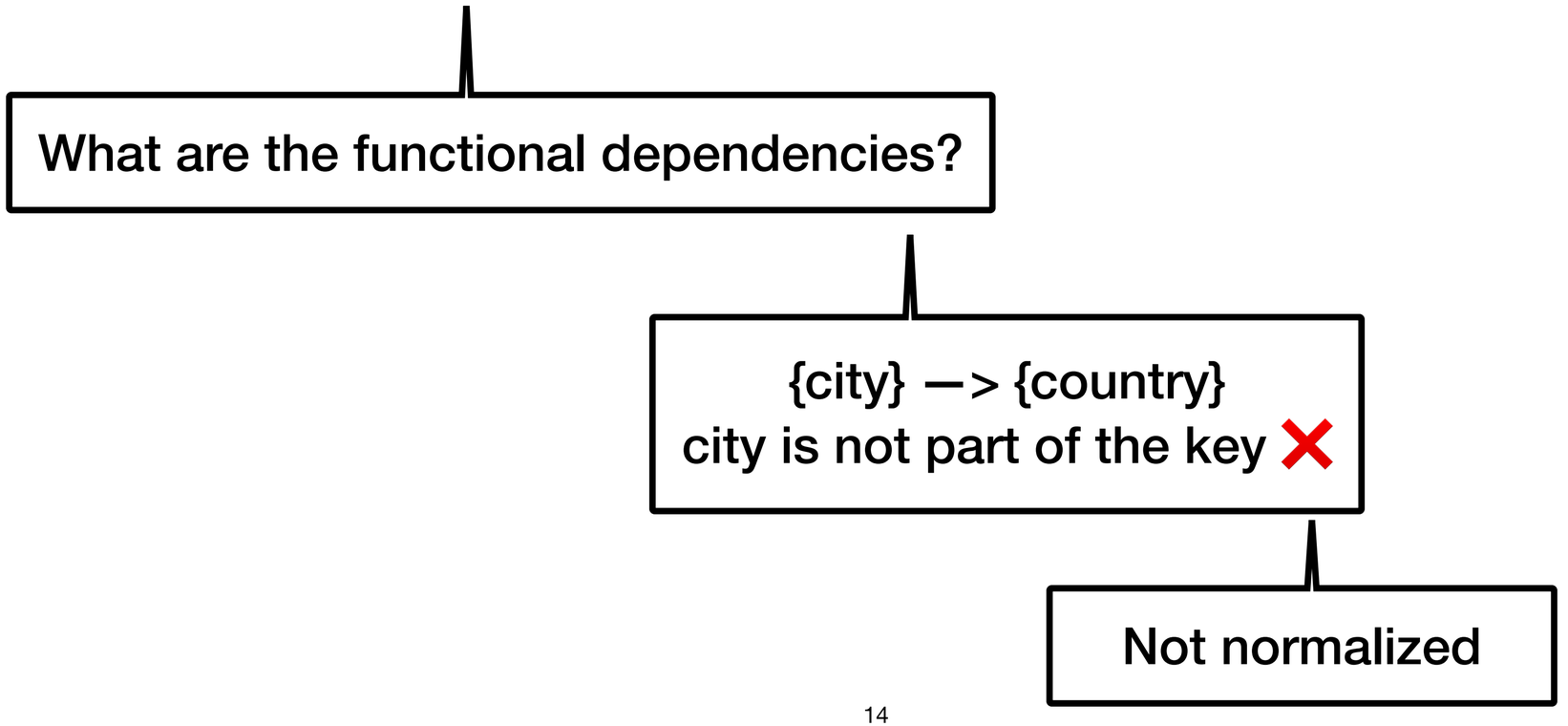
What are the functional dependencies?

{city} —> {country}
city is not part of the key ❌

Normalization - Example

users-ver1

user_id	fname	lname	city	country
101	Rubi	Boim	Tel Aviv	Israel
102	Tova	Milo	Tel Aviv	Israel
103	Lebron	James	Los Angeles	USA



Normalization - Example

users-ver1

user_id	fname	lname	city	country
101	Rubi	Boim	Tel Aviv	Israel
102	Tova	Milo	Tel Aviv	Israel
103	Lebron	James	Los Angeles	USA

What are the functional dependencies?

{city} —> {country}
city is not part of the key ❌

Not normalized

So how can you normalize it?

Normalization - Example

users-ver1

user_id	fname	lname	city	country
101	Rubi	Boim	Tel Aviv	Israel
102	Tova	Milo	Tel Aviv	Israel
103	Lebron	James	Los Angeles	USA

users-ver2

user_id	fname	lname	city
101	Rubi	Boim	Tel Aviv
102	Tova	Milo	Tel Aviv
103	Lebron	James	Los Angeles

cities

city	country
Tel Aviv	Israel
Los Angeles	USA

Normalization - Example

users-ver1

user_id	fname	lname	city	country
101	Rubi	Boim	Tel Aviv	Israel
102	Tova	Milo	Tel Aviv	Israel
103	Lebron	James	Los Angeles	USA

We assume users will be much bigger than cities —> reduced storage

What is the difference? Data integrity? Storage?

users-ver2

user_id	fname	lname	city
101	Rubi	Boim	Tel Aviv
102	Tova	Milo	Tel Aviv
103	Lebron	James	Los Angeles

cities

city	country
Tel Aviv	Israel
Los Angeles	USA

Normalized DB

- Data integrity ✔
- Reduced Storage ✔
- JOINS are used to retrieve data

```
SELECT users.*, cities.country
FROM users, cities
WHERE users.city = cities.city
```

users

user_id	fname	lname	city
101	Rubi	Boim	Tel Aviv
102	Tova	Milo	Tel Aviv
103	Lebron	James	Los Angeles

cities

city	country
Tel Aviv	Israel
Los Angeles	USA

Normalized DB

So whats the problem in Big Data?

- Data integrity ✔
- Reduced Storage ✔
- JOINS are used to retrieve data

```
SELECT users.*, cities.country
FROM users, cities
WHERE users.city = cities.city
```

users

user_id	fname	lname	city
101	Rubi	Boim	Tel Aviv
102	Tova	Milo	Tel Aviv
103	Lebron	James	Los Angeles

cities

city	country
Tel Aviv	Israel
Los Angeles	USA

Normalized DB

So whats the problem in Big Data?

- Data integrity ✓ — do not scale
- Reduced Storage ✓ — cheap today
- JOINS are used to retrieve data — do not scale

```
SELECT users.*, cities.country
FROM users, cities
WHERE users.city = cities.city
```

users

user_id	fname	lname	city
101	Rubi	Boim	Tel Aviv
102	Tova	Milo	Tel Aviv
103	Lebron	James	Los Angeles

cities

city	country
Tel Aviv	Israel
Los Angeles	USA

Denormalization

- No 3NF
- Improve reads / writes performance dramatically
  - At the expense of data integrity and storage
  - Requires more writes
- Also applied on relational DBs
- “Workaround” when joins are unavailable (wide columns DBs)

Example (continue)

cities

city	country
Tel Aviv	Israel
Los Angeles	USA

```
SELECT users.*, cities.country
FROM users, cities
WHERE users.city = cities.city
```

users

user_id	fname	lname	city
101	Rubi	Boim	Tel Aviv
102	Tova	Milo	Tel Aviv
103	Lebron	James	Los Angeles

```
SELECT * FROM users-deno
```

users-deno

user_id	fname	lname	city	country
101	Rubi	Boim	Tel Aviv	Israel
102	Tova	Milo	Tel Aviv	Israel
103	Lebron	James	Los Angeles	USA

Example (continue)

cities

city	country
Tel Aviv	Israel
Los Angeles	USA

```
SELECT users.*, cities.country
FROM users, cities
WHERE users.city = cities.city
```

users

user_id	fname	lname	city
101	Rubi	Boim	Tel Aviv
102	Tova	Milo	Tel Aviv
103	Lebron	James	Los Angeles

```
SELECT * FROM users-deno
```

users-deno

user_id	fname	lname	city	country
101	Rubi	Boim	Tel Aviv	Israel
102	Tova	Milo	Tel Aviv	Israel
103	Lebron	James	Los Angeles	USA

Which version is faster?

Example (continue)

cities

city	country
Tel Aviv	Israel
Los Angeles	USA

```
SELECT users.*, cities.country
FROM users, cities
WHERE users.city = cities.city
```

users

user_id	fname	lname	city
101	Rubi	Boim	Tel Aviv
102	Tova	Milo	Tel Aviv
103	Lebron	James	Los Angeles

```
SELECT * FROM users-deno
```

users-deno

user_id	fname	lname	city	country
101	Rubi	Boim	Tel Aviv	Israel
102	Tova	Milo	Tel Aviv	Israel
103	Lebron	James	Los Angeles	USA

Which version is faster?

Example (continue)

cities

city	country
Tel Aviv	Israel
Los Angeles	USA

```
SELECT users.*, cities.country
FROM users, cities
WHERE users.city = cities.city
```

users

user_id	fname	lname	city
101	Rubi	Boim	Tel Aviv
102	Tova	Milo	Tel Aviv
103	Lebron	James	Los Angeles

```
SELECT * FROM users-deno
```

users-deno

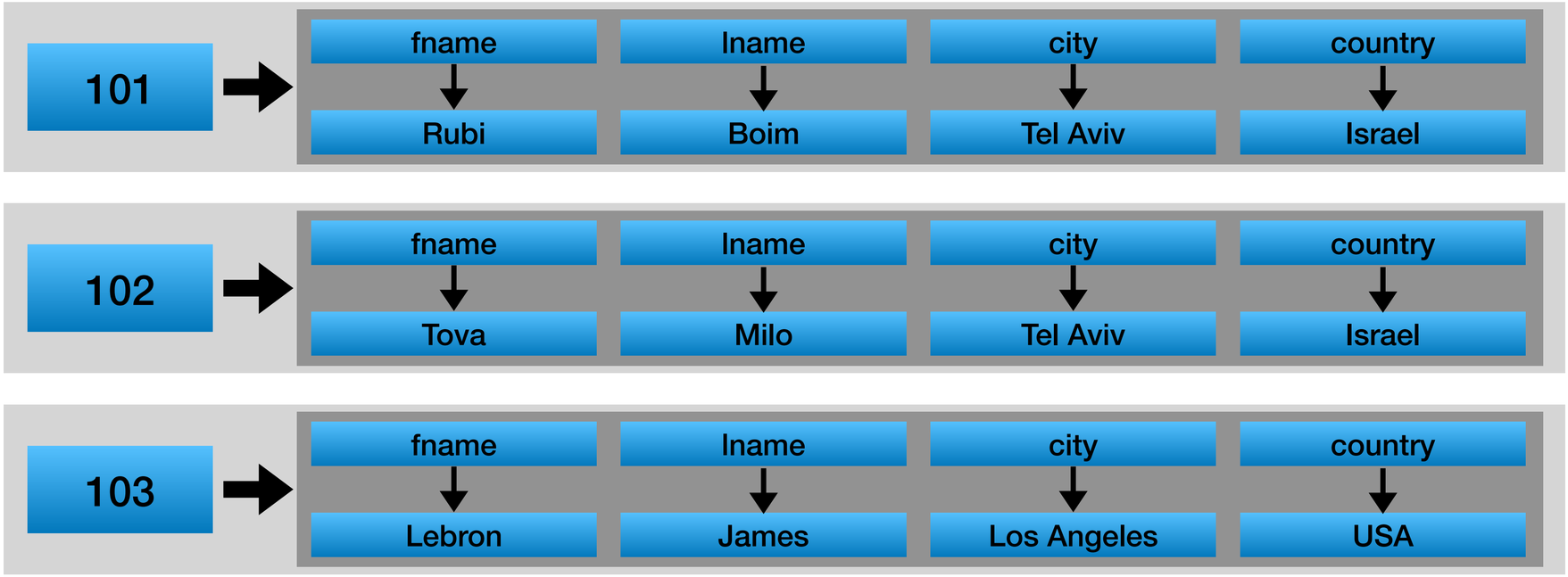
user_id	fname	lname	city	country
101	Rubi	Boim	Tel Aviv	Israel
102	Tova	Milo	Tel Aviv	Israel
103	Lebron	James	Los Angeles	USA

Which version is faster?

If we have 10k queries per second, which will be (much) faster?

Moving to Cassandra

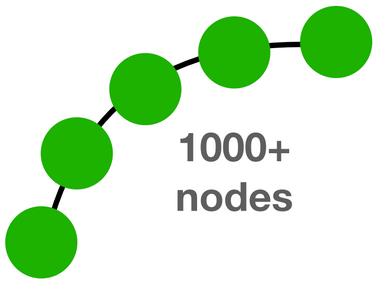
user_id	fname	lname	city	country
101	Rubi	Boim	Tel Aviv	Israel
102	Tova	Milo	Tel Aviv	Israel
103	Lebron	James	Los Angeles	USA



What about this query

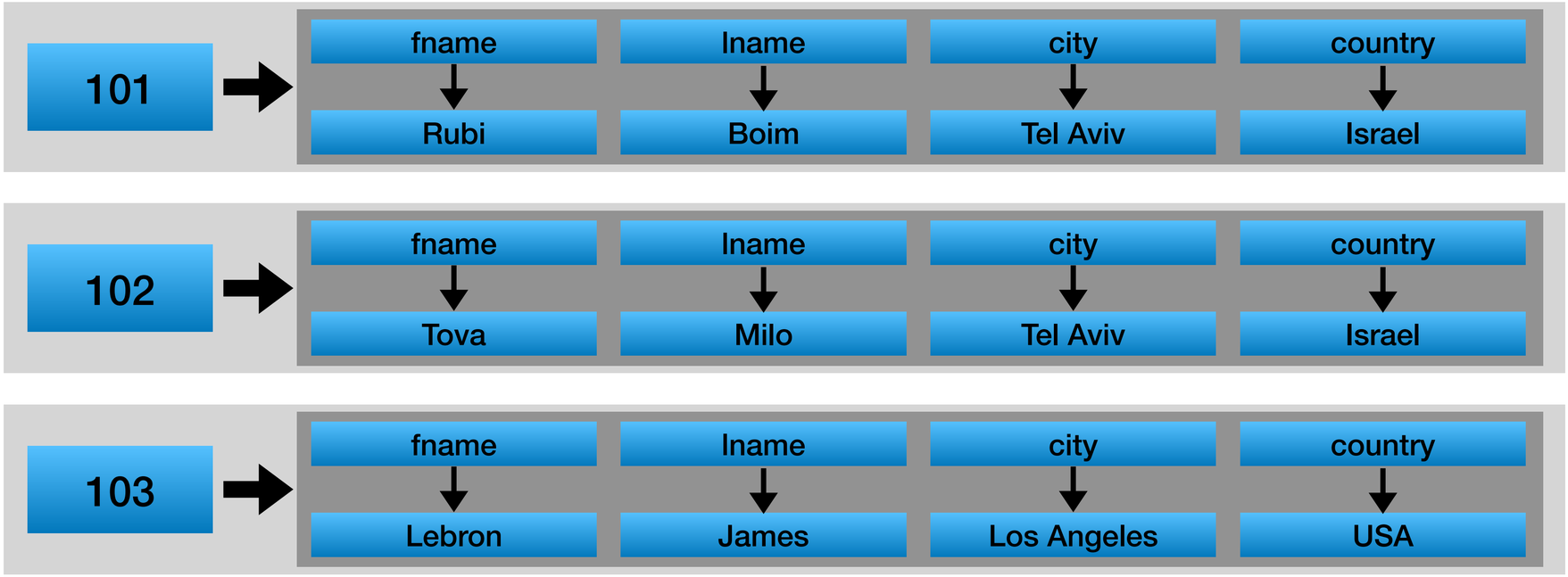
```
SELECT * FROM users
WHERE country = "Israel"
```

Is this an efficient query?



Moving to Cassandra

user_id	fname	lname	city	country
101	Rubi	Boim	Tel Aviv	Israel
102	Tova	Milo	Tel Aviv	Israel
103	Lebron	James	Los Angeles	USA

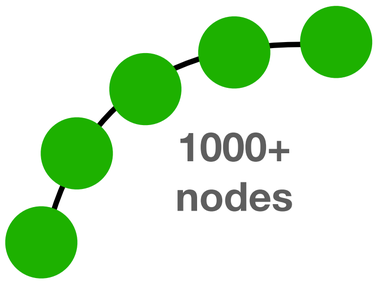


What about this query

```
SELECT * FROM users
WHERE country = "Israel"
```

Is this an efficient query?

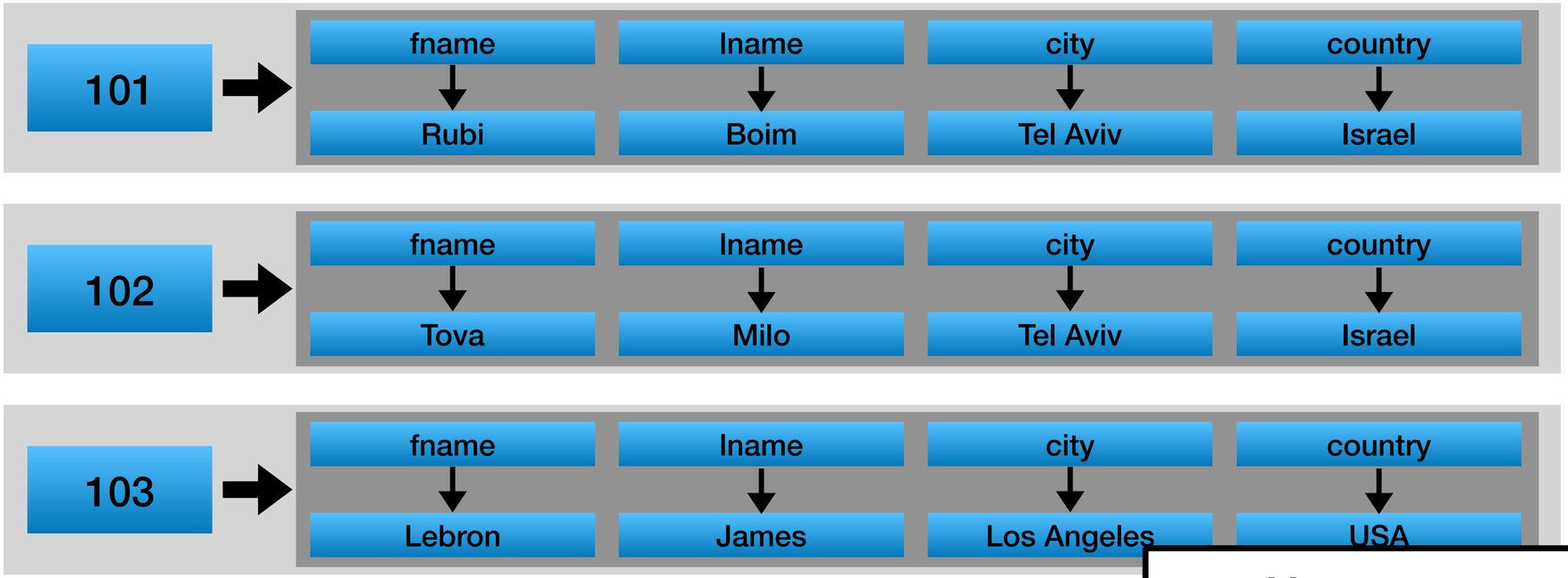
No



Error! It is not even a valid query…

Moving to Cassandra

user_id	fname	lname	city	country
101	Rubi	Boim	Tel Aviv	Israel
102	Tova	Milo	Tel Aviv	Israel
103	Lebron	James	Los Angeles	USA



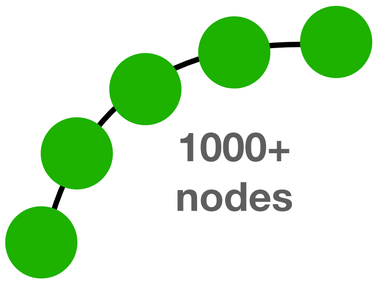
How can we solve this?

What about this query

```
SELECT * FROM users
WHERE country = "Israel"
```

Is this an efficient query?

No



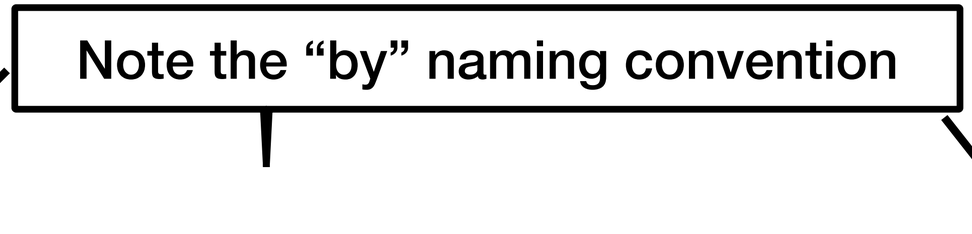
Error! It is not even a valid query…

Denormalization by 2 tables

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

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

Denormalization by 2 tables




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

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

Denormalization by 2 tables

Note the “by” naming convention

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

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

Would this work?

Denormalization by 2 tables

Note the “by” naming convention

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

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

Would this work?

How many rows do we have here?

Denormalization by 2 tables

Note the “by” naming convention

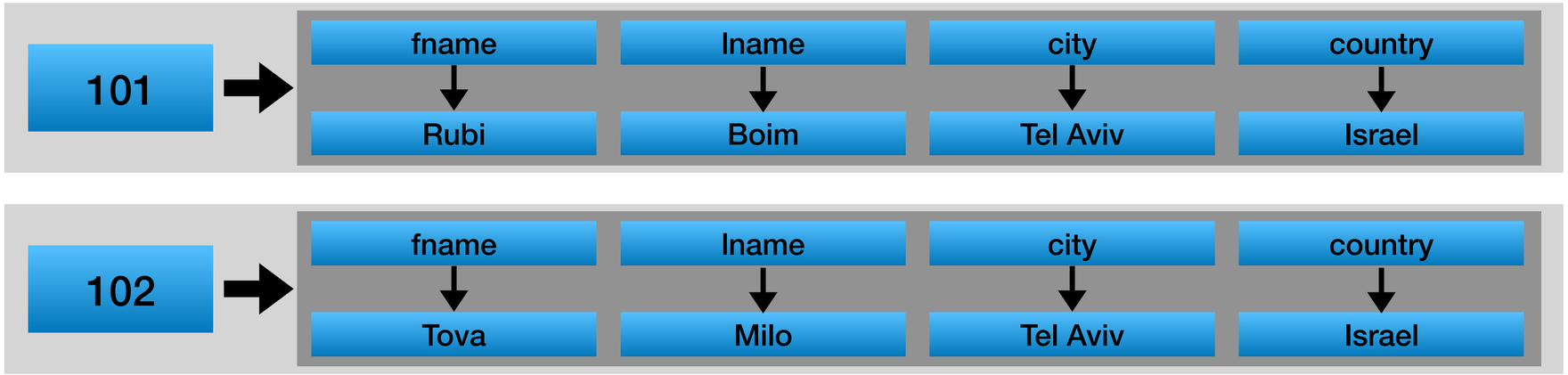
```
CREATE TABLE users_by_id (
   user_id      BIGINT,
   fname        TEXT,
   lname        TEXT,
   city         TEXT,
   country      TEXT,
   PRIMARY KEY (user_id)
);
```

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

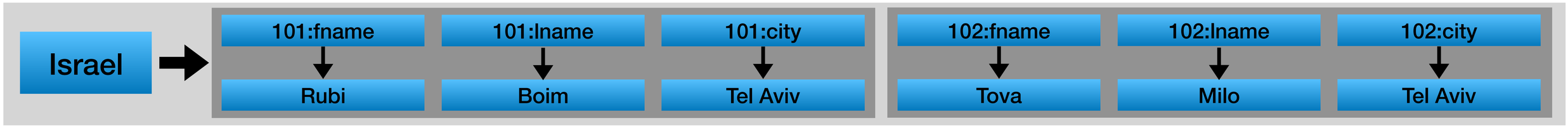
Now it is ok :)

Denormalization by 2 tables

`users_by_id`

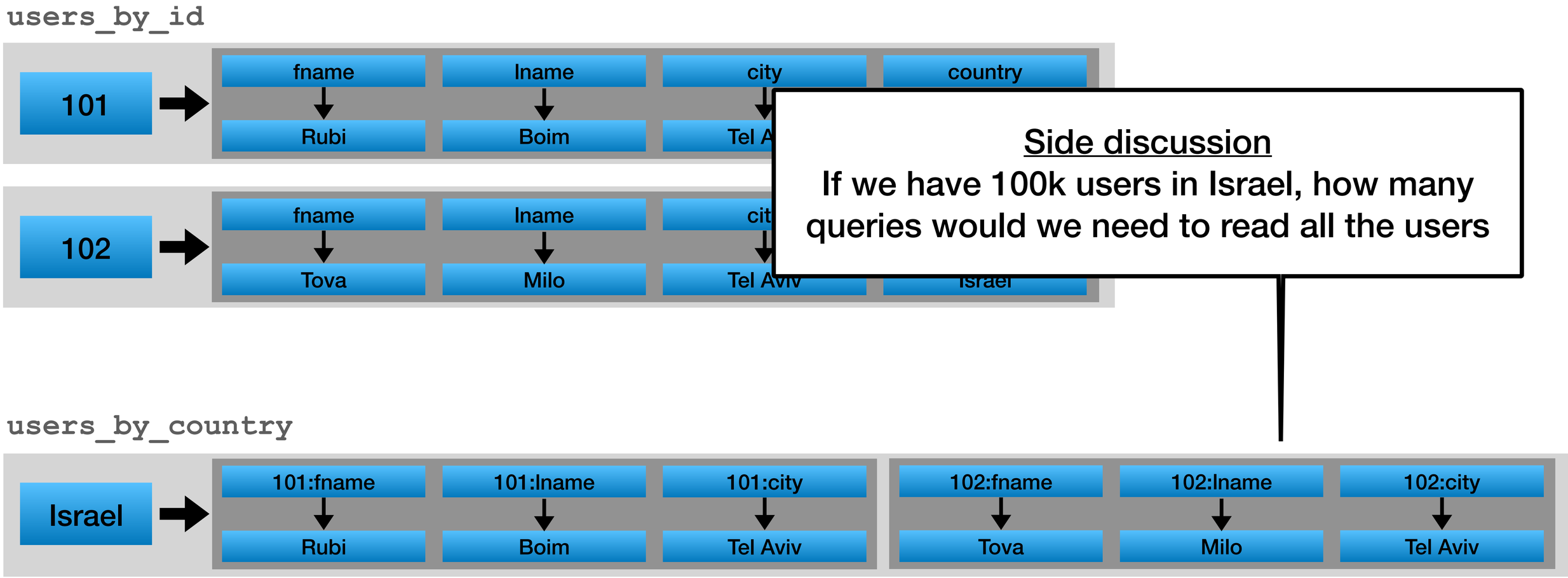


`users_by_country`



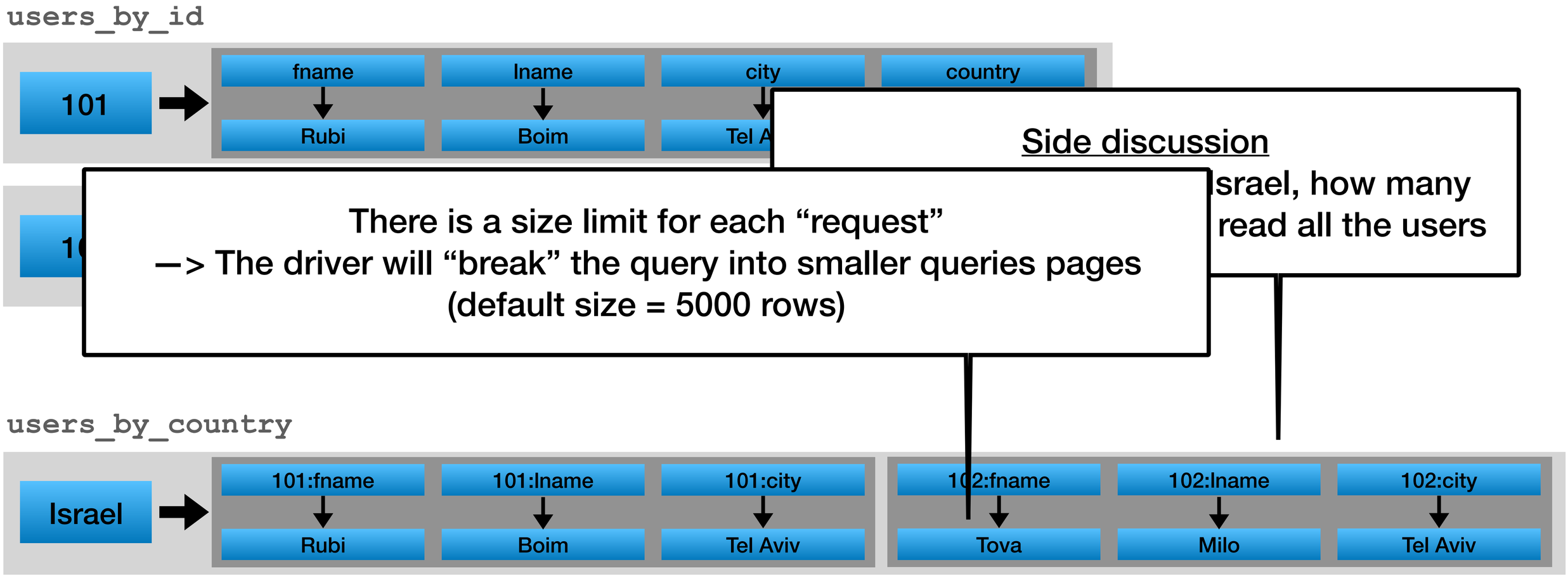

```
SELECT * FROM users_by_country WHERE country = "Israel"
```

Denormalization by 2 tables




```
SELECT * FROM users_by_country WHERE country = "Israel"
```

Denormalization by 2 tables



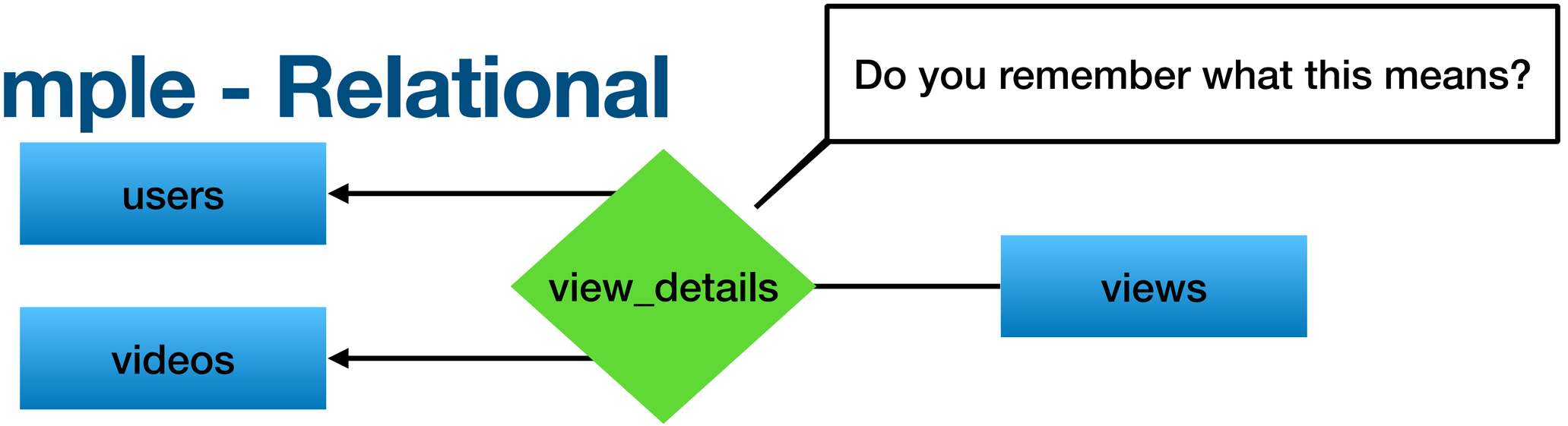

```
SELECT * FROM users_by_country WHERE country = “Israel”
```

Interesting .

- We saw 2 different ways to denormalize:
  - Merging 2 tables into 1 table
  - Splitting 1 table into 2 tables

- Denormalization is crucial for “correct” data modeling in Big Data

Example - Relational



users

user_id	name	city	…
101	Rubi Boim	Tel Aviv	
102	Tova Milo	Tel Aviv	
103	Lebron James	Los Angeles	

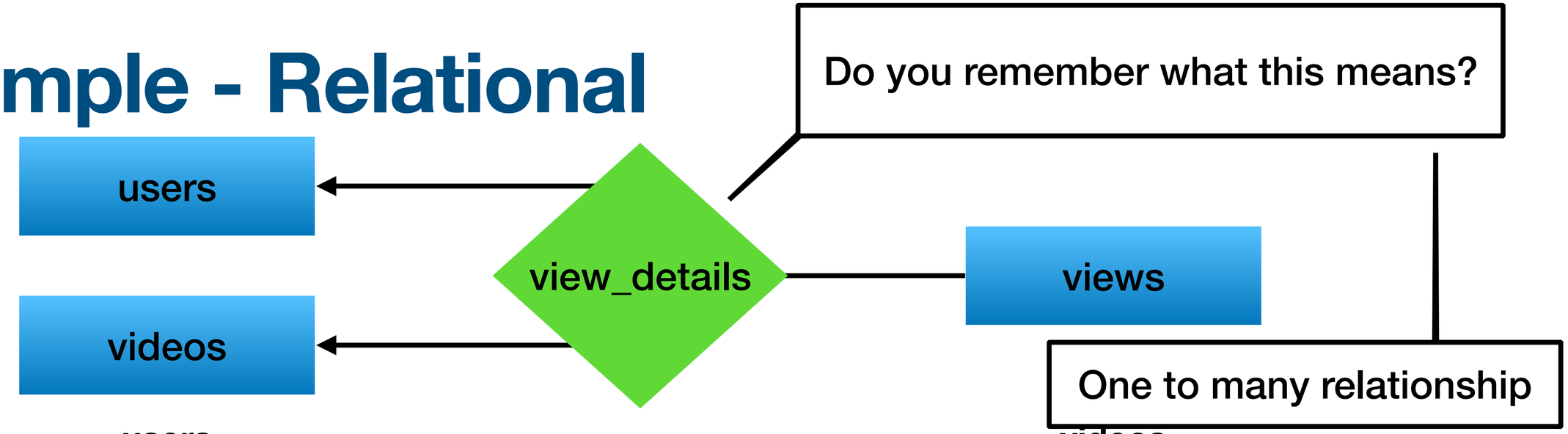
videos

video_id	title	year	…
1	Bad Boys	1995	
2	Top Gun	1986	
3	American Pie	1999	

views

view_id	user_id	video_id	device	…
2382	101	1	Apple TV	
2383	101	2	Apple TV	
2384	102	1	iPhone	

Example - Relational



users

user_id	name	city	…
101	Rubi Boim	Tel Aviv	
102	Tova Milo	Tel Aviv	
103	Lebron James	Los Angeles	

videos

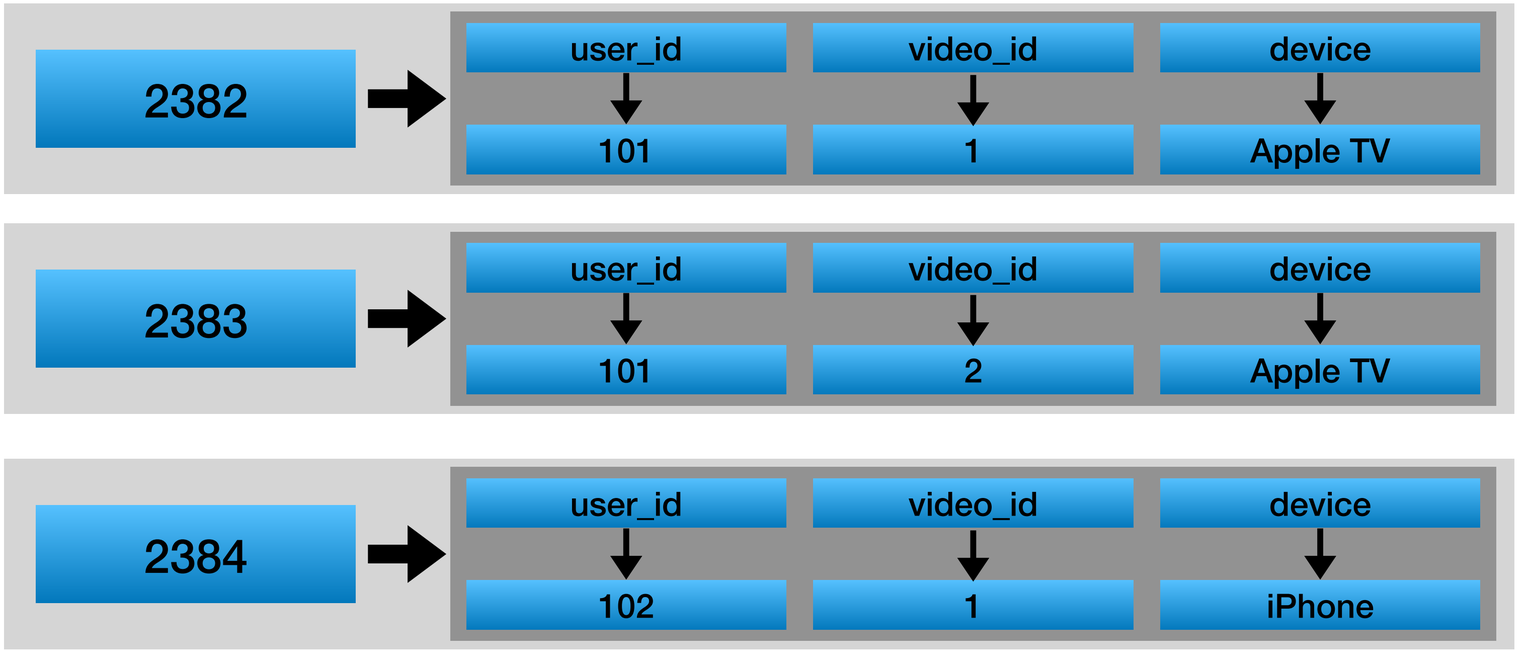
video_id	title	year	…
1	Bad Boys	1995	
2	Top Gun	1986	
3	American Pie	1999	

views

view_id	user_id	video_id	device	…
2382	101	1	Apple TV	
2383	101	2	Apple TV	
2384	102	1	iPhone	

Example - Cassandra

```
CREATE TABLE views (
    view_id         TIMEUUID,
    user_id         BIGINT,
    video_id        BIGINT,
    device          TEXT,
    PRIMARY KEY ((view_id))
);
```



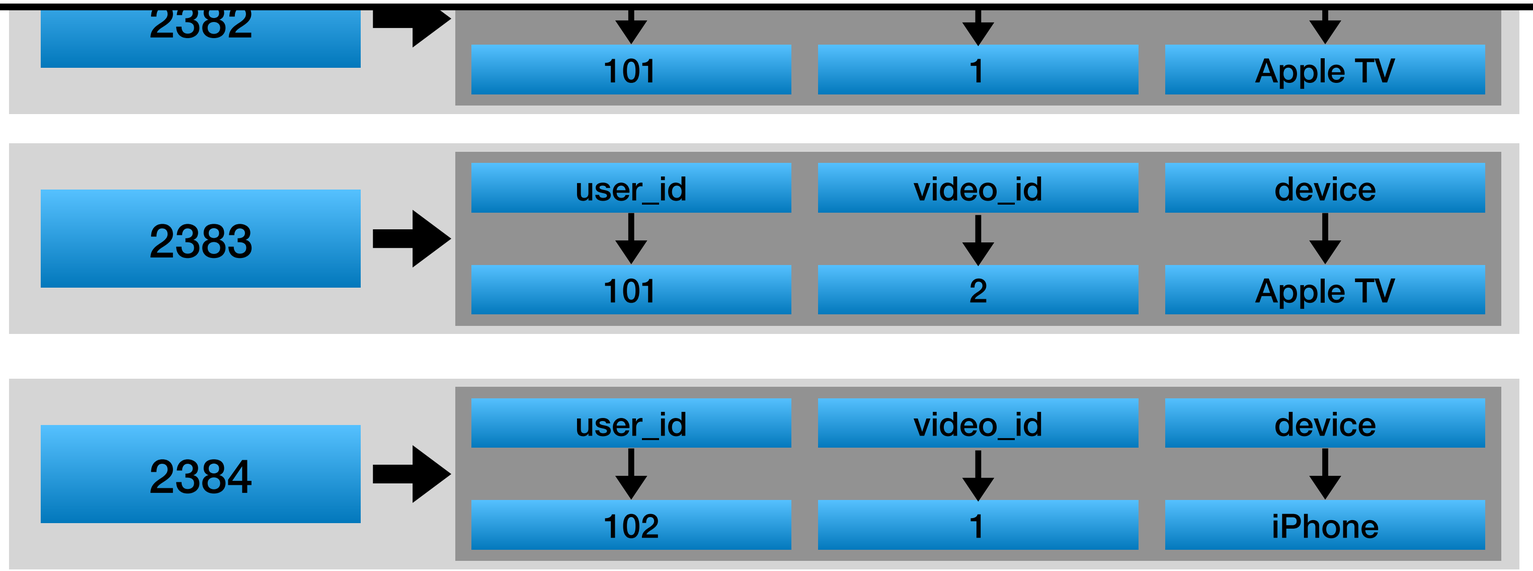
views

view_id	user_id	video_id	device	...
2382	101	1	Apple TV	
2383	101	2	Apple TV	
2384	102	1	iPhone	

Example - Cassandra

Side discussion - why did we switch to TIMEUUID instead of Integer?

```
CREATE TABLE views (
    view_id         TIMEUUID,
    user_id         BIGINT,
    video_id        BIGINT,
    device          TEXT,
    PRIMARY KEY ((view_id))
);
```



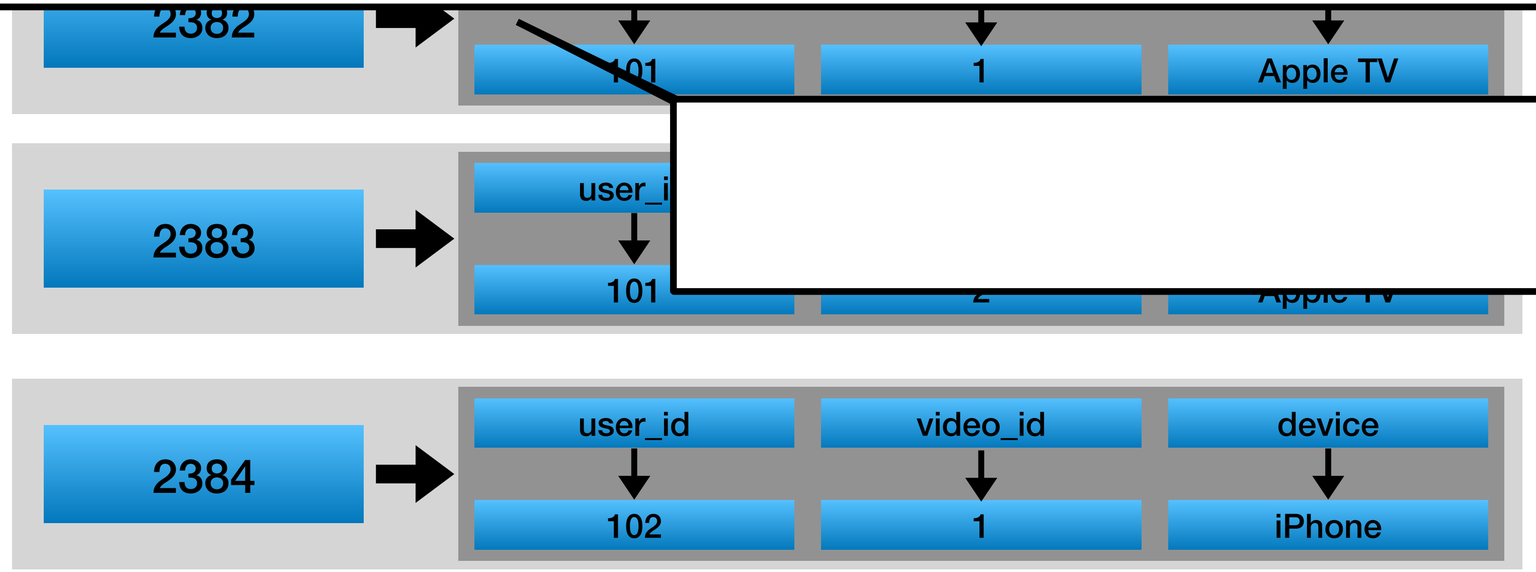
views

view_id	user_id	video_id	device	...
2382	101	1	Apple TV	
2383	101	2	Apple TV	
2384	102	1	iPhone	

Example - Cassandra

```
CREATE TABLE views (
    view_id         TIMEUUID,
    user_id         BIGINT,
    video_id        BIGINT,
    device          TEXT,
    PRIMARY KEY ((view_id))
);
```

Side discussion - why did we switch to TIMEUUID instead of Integer?

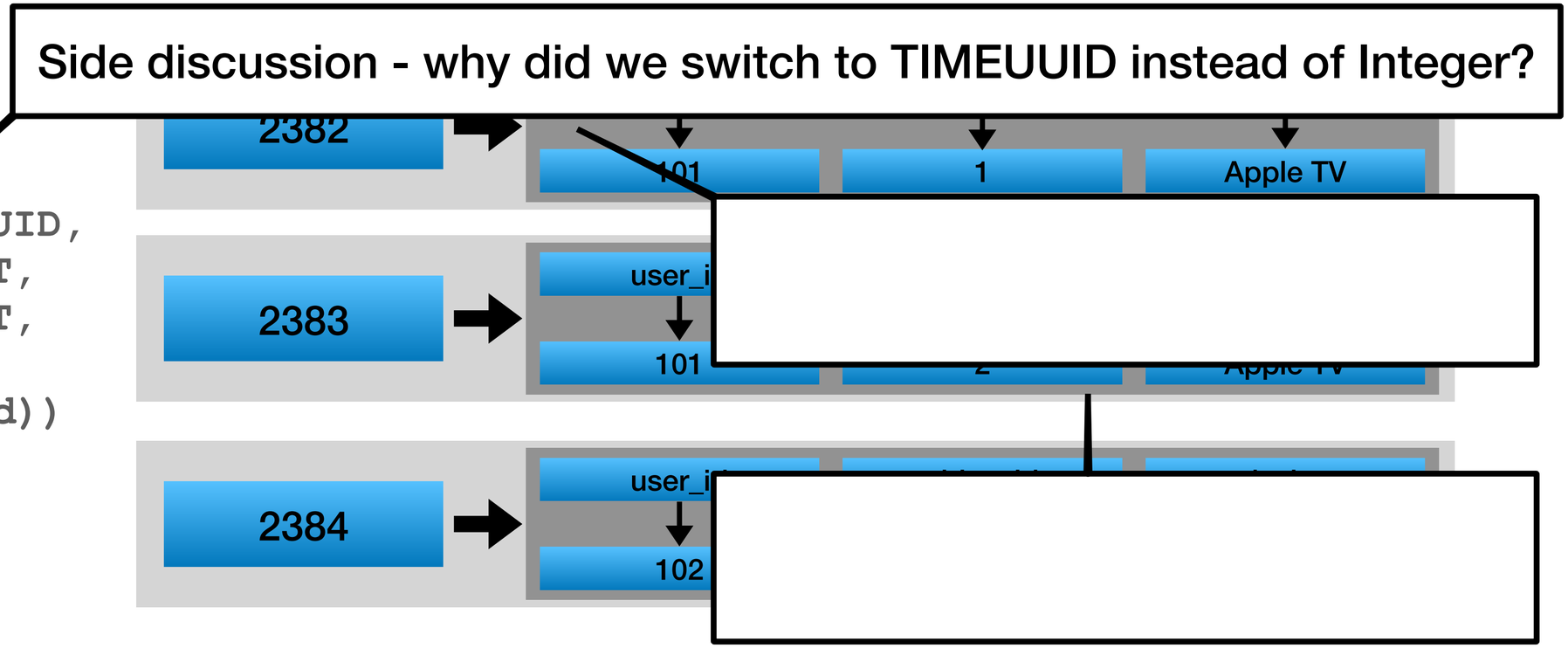


views

view_id	user_id	video_id	device	...
2382	101	1	Apple TV	
2383	101	2	Apple TV	
2384	102	1	iPhone	

Example - Cassandra

```
CREATE TABLE views (
    view_id         TIMEUUID,
    user_id         BIGINT,
    video_id        BIGINT,
    device          TEXT,
    PRIMARY KEY ((view_id))
);
```

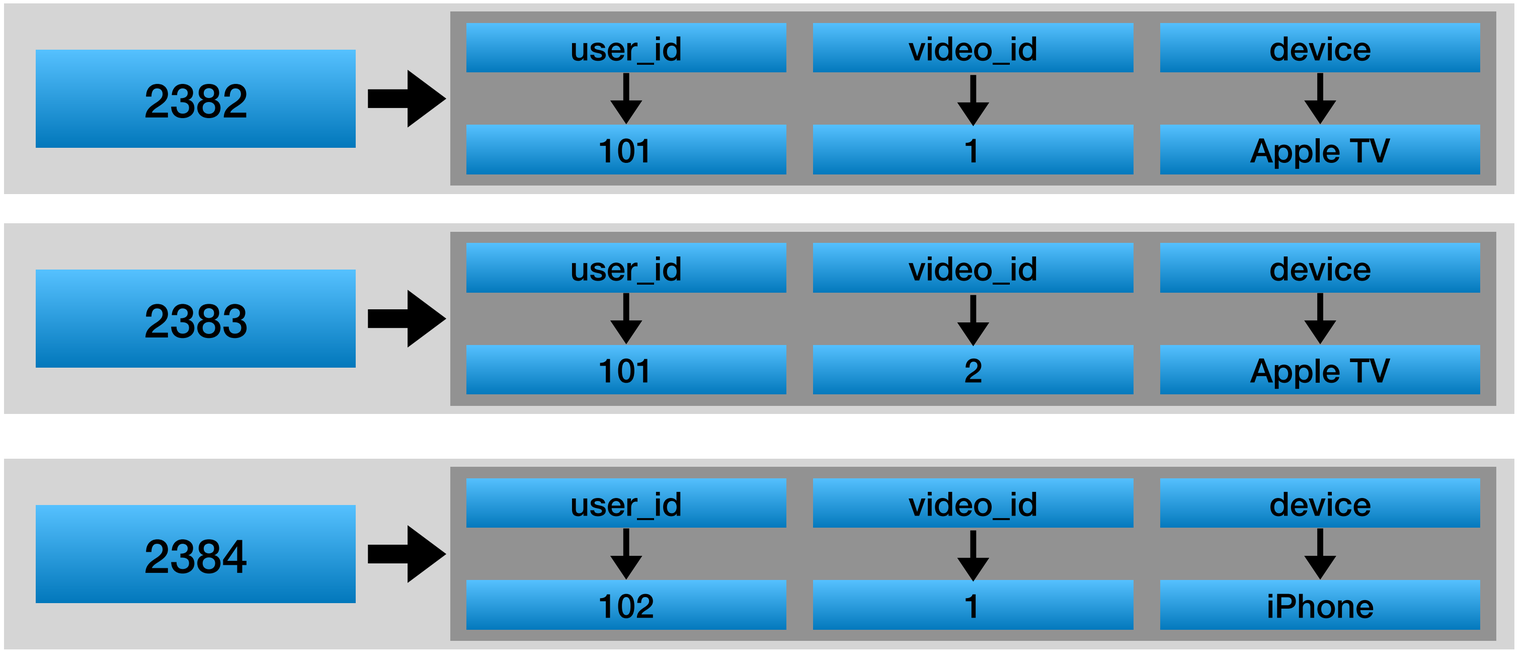


views

view_id	user_id	video_id	device	...
2382	101	1	Apple TV	
2383	101	2	Apple TV	
2384	102	1	iPhone	

Example - Cassandra

```
CREATE TABLE views (
    view_id         TIMEUUID,
    user_id         BIGINT,
    video_id        BIGINT,
    device          TEXT,
    PRIMARY KEY ((view_id))
);
```



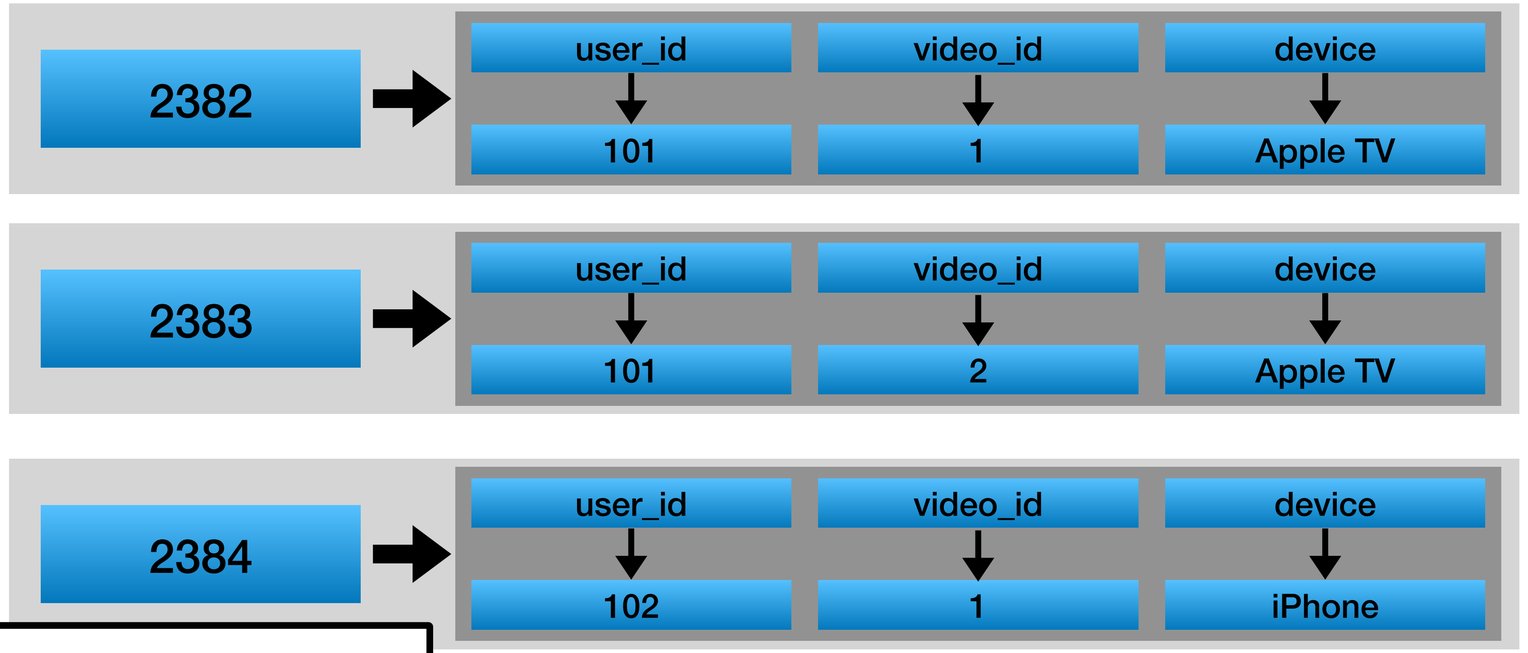
Which queries can we efficiently return?

views

view_id	user_id	video_id	device	...
2382	101	1	Apple TV	
2383	101	2	Apple TV	
2384	102	1	iPhone	

Example - Cassandra

```
CREATE TABLE views (
    view_id         TIMEUUID,
    user_id         BIGINT,
    video_id        BIGINT,
    device          TEXT,
    PRIMARY KEY ((view_id))
);
```



SELECT * FROM views WHERE view_id = ?

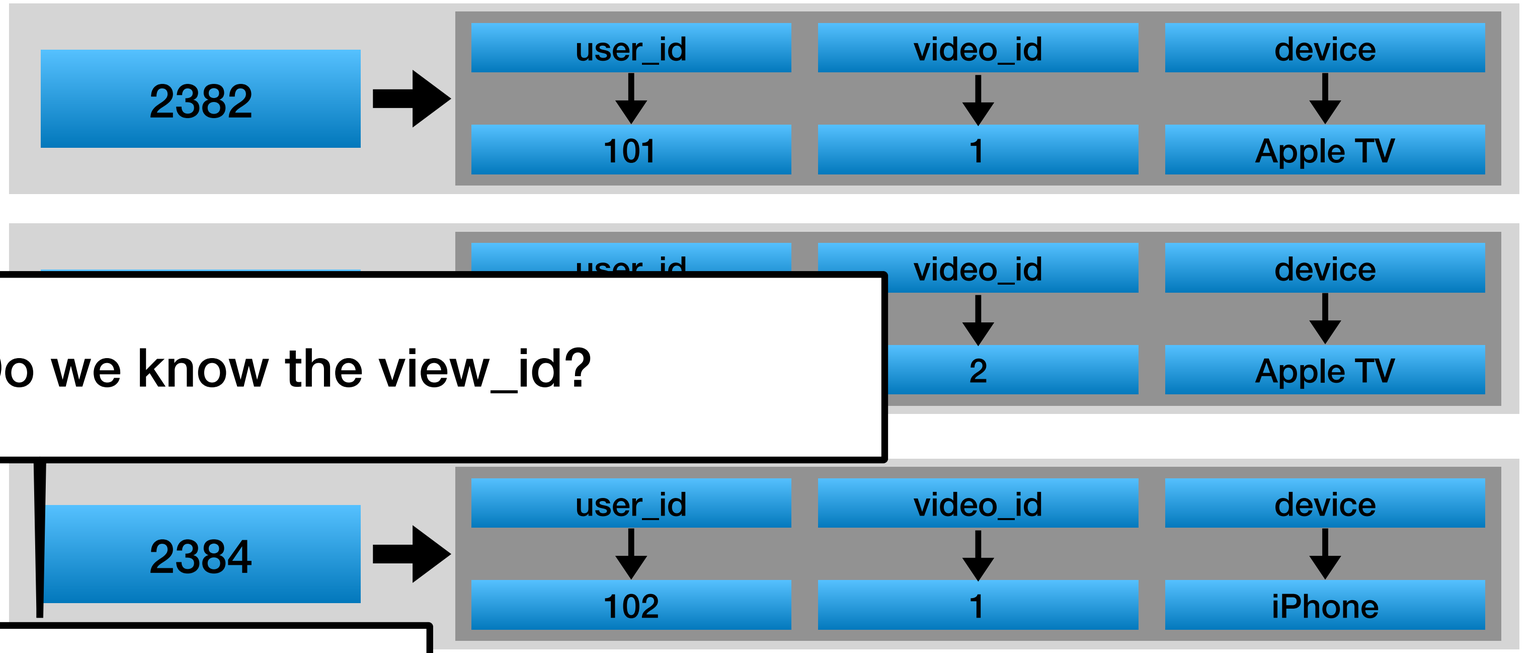
Which queries can we efficiently return?

views

view_id	user_id	video_id	device	...
2382	101	1	Apple TV	
2383	101	2	Apple TV	
2384	102	1	iPhone	

Example - Cassandra

```
CREATE TABLE views (
    view_id         TIMEUUID,
    user_id         BIGINT
    video_id
    device
    PRIMARY KEY ((
);
```



Do we know the view_id?

SELECT * FROM views WHERE view_id = ?

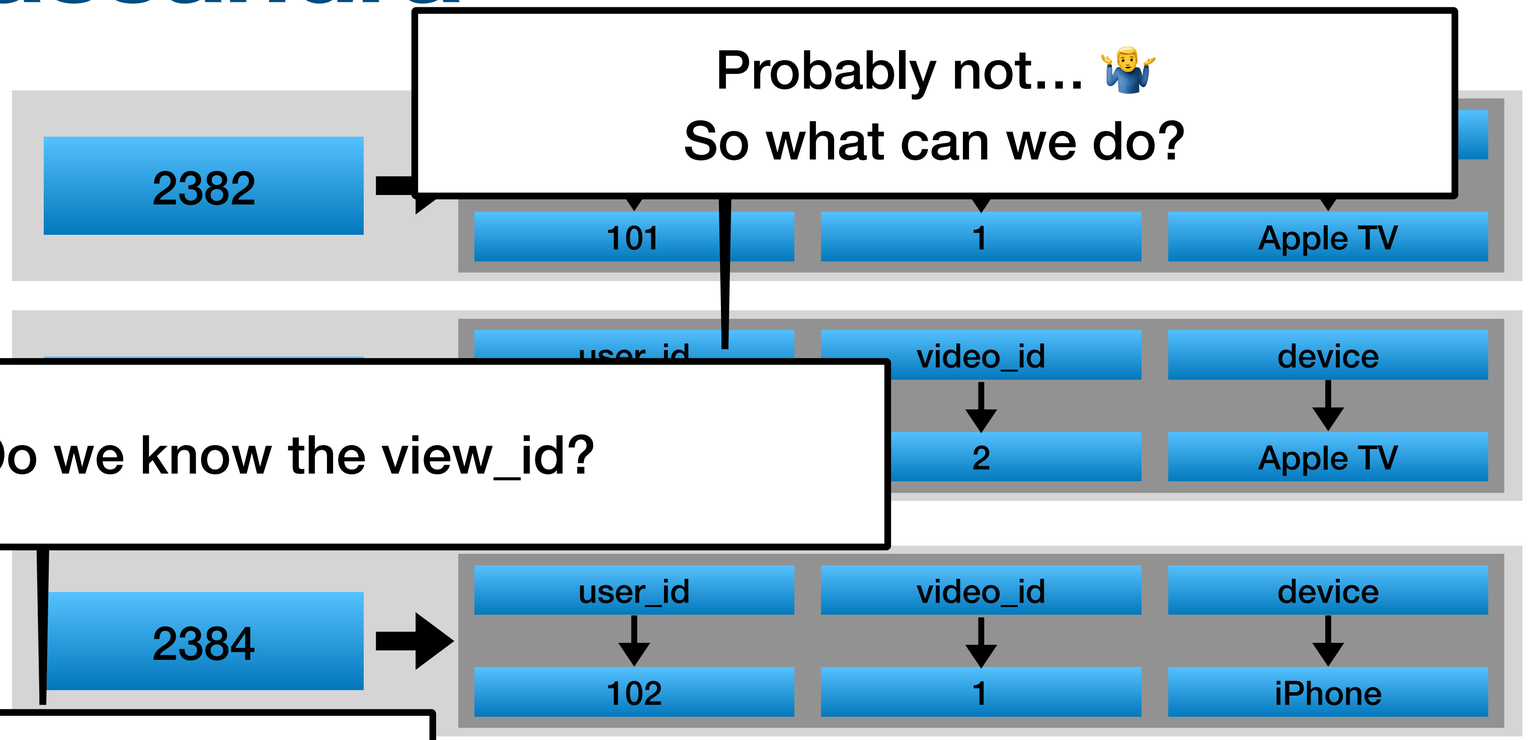
Which queries can we efficiently return?

views

view_id	user_id	video_id	device	…
2382	101	1	Apple TV	
2383	101	2	Apple TV	
2384	102	1	iPhone	

Example - Cassandra

```
CREATE TABLE views (
    view_id         TIMEUUID,
    user_id         BIGINT
    video_id
    device
    PRIMARY KEY ((
);
```



SELECT * FROM views WHERE view_id = ?

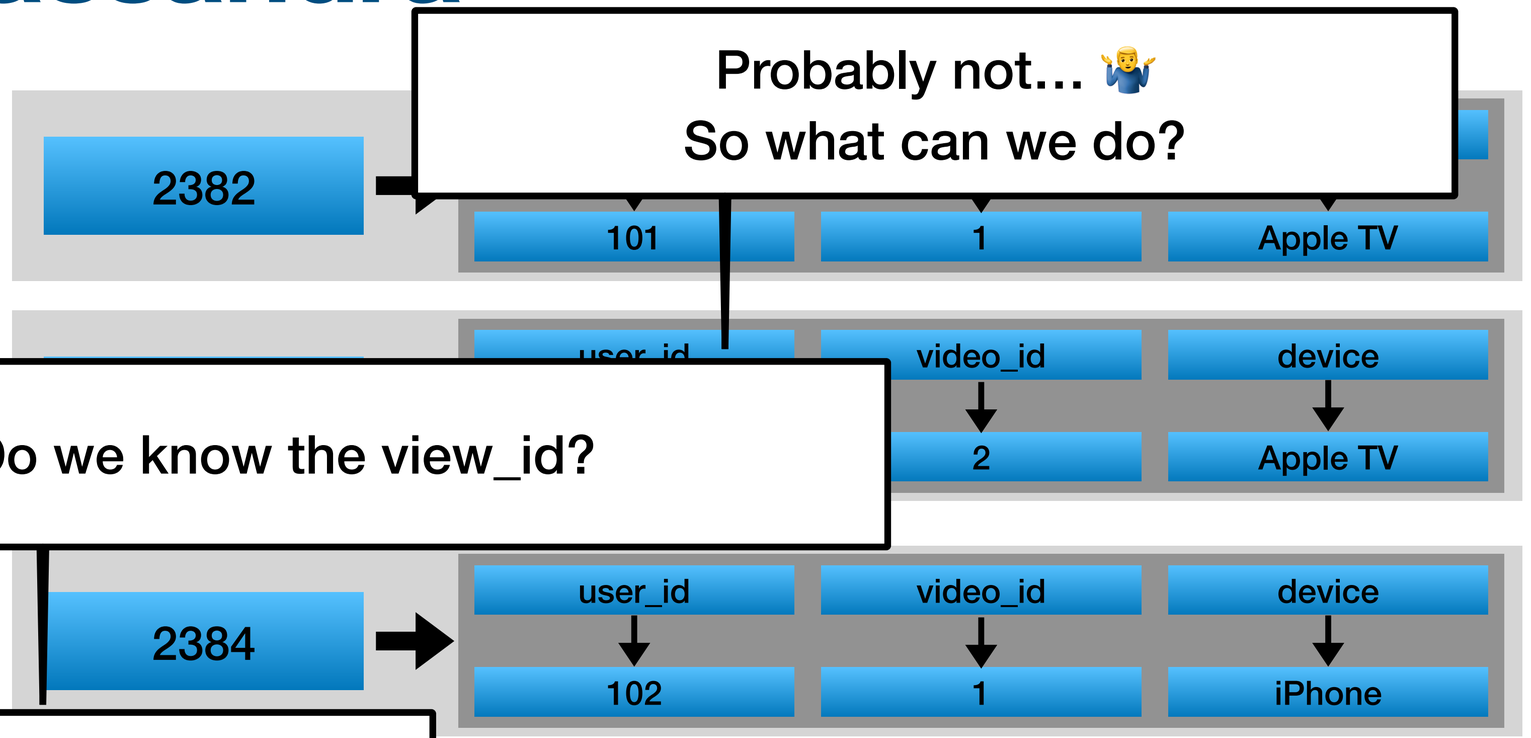
Which queries can we efficiently return?

views

view_id	user_id	video_id	device	…
2382	101	1	Apple TV	
2383	101	2	Apple TV	
2384	102	1	iPhone	

Example - Cassandra

```
CREATE TABLE views (
    view_id         TIMEUUID,
    user_id         BIGINT
    video_id
    device
    PRIMARY KEY ((
);
```



SELECT * FROM views WHERE view_id = ?

Which queries can we efficiently return?

views

view_id	user_id	video_id	device	…

Probably the queries will be by the user/movie
How can we support this?

Denormalization by 2 tables

```
CREATE TABLE views_by_user (
    user_id         BIGINT,
    view_id         TIMEUUID,
    video_id        BIGINT,
    device          TEXT,
    PRIMARY KEY ((user_id), view_id)
) WITH CLUSTERING ORDER BY
  (view_id DESC);
```

```
CREATE TABLE views_by_video (
    video_id        BIGINT,
    view_id         TIMEUUID,
    user_id         BIGINT,
    device          TEXT,
    PRIMARY KEY ((video_id), view_id)
) WITH CLUSTERING ORDER BY
  (view_id DESC);
```

Any efficient queries?

Denormalization by 2 tables

```
CREATE TABLE views_by_user (
    user_id         BIGINT,
    view_id         TIMEUUID,
    video_id        BIGINT,
    device          TEXT,
    PRIMARY KEY ((user_id), view_id)
) WITH CLUSTERING ORDER BY
  (view_id DESC);
```

```
CREATE TABLE views_by_video (
    video_id        BIGINT,
    view_id         TIMEUUID,
    user_id         BIGINT,
    device          TEXT,
    PRIMARY KEY ((video_id), view_id)
) WITH CLUSTERING ORDER BY
  (view_id DESC);
```

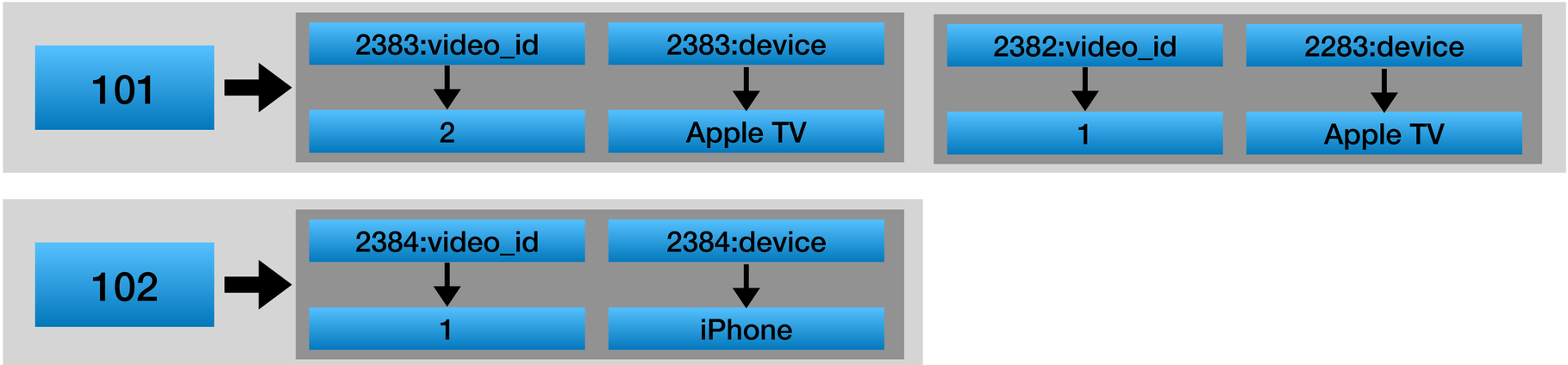
Any efficient queries?

get all recent views of a user

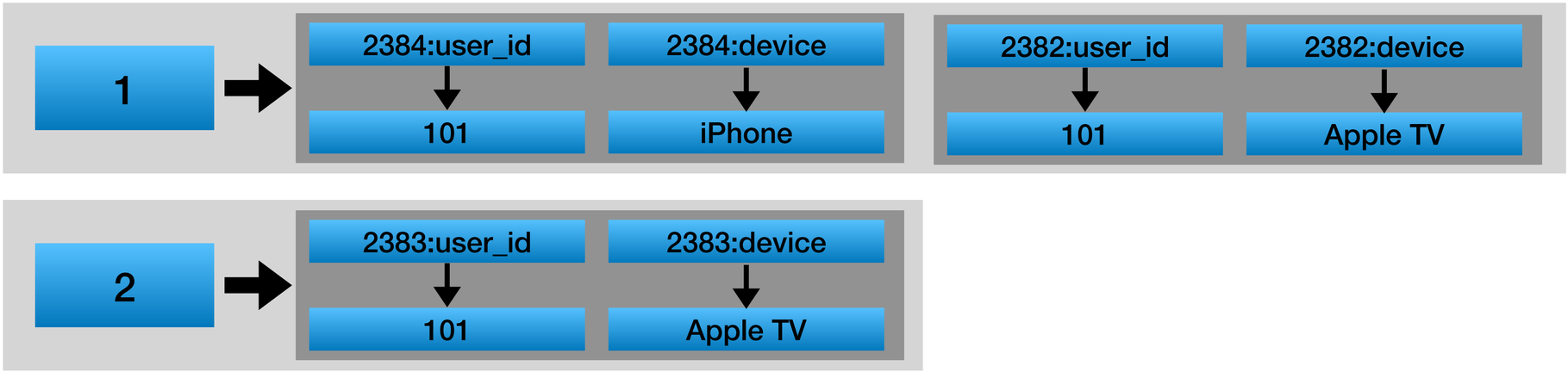
get all recent views of a video

Denormalization by 2 tables

views_by_user



views_by_video



Example continue

Why do we care?

How can we return all the views for a specific day?

```
CREATE TABLE views_by_user (
    user_id         BIGINT,
    view_id         TIMEUUID,
    video_id        BIGINT,
    device          TEXT,
    PRIMARY KEY ((user_id), view_id)
) WITH CLUSTERING ORDER BY
  (view_id DESC);
```

```
CREATE TABLE views_by_video (
    video_id        BIGINT,
    view_id         TIMEUUID,
    user_id         BIGINT,
    device          TEXT,
    PRIMARY KEY ((video_id), view_id)
) WITH CLUSTERING ORDER BY
  (view_id DESC);
```

Example continue

Why do we care?

Calculating trending videos for example

How can we return all the views for a specific day?

```
CREATE TABLE views_by_user (
    user_id         BIGINT,
    view_id         TIMEUUID,
    video_id        BIGINT,
    device          TEXT,
    PRIMARY KEY ((user_id), view_id)
) WITH CLUSTERING ORDER BY
  (view_id DESC);
```

```
CREATE TABLE views_by_video (
    video_id        BIGINT,
    view_id         TIMEUUID,
    user_id         BIGINT,
    device          TEXT,
    PRIMARY KEY ((video_id), view_id)
) WITH CLUSTERING ORDER BY
  (view_id DESC);
```

Example continue

How can we return all the views for a specific day?

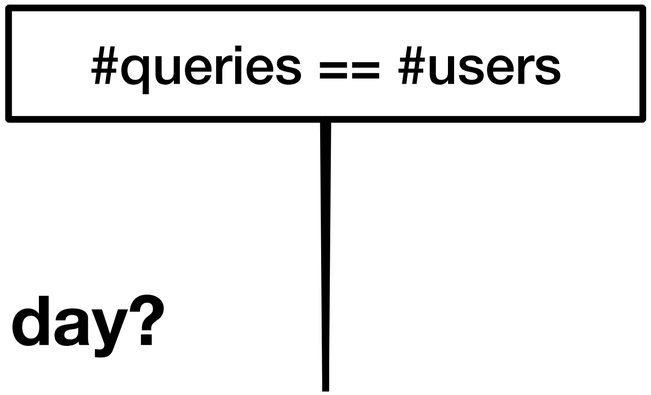
- Create a job (Spark?) that reads all the views from views_by_user and filter the result

- Create a job (Spark?) that reads all the views from views_by_video and filter the result

- Denormalize to another table

What is the difference between the options (time / IO)?

Example continue



How can we return all the views for a specific day?

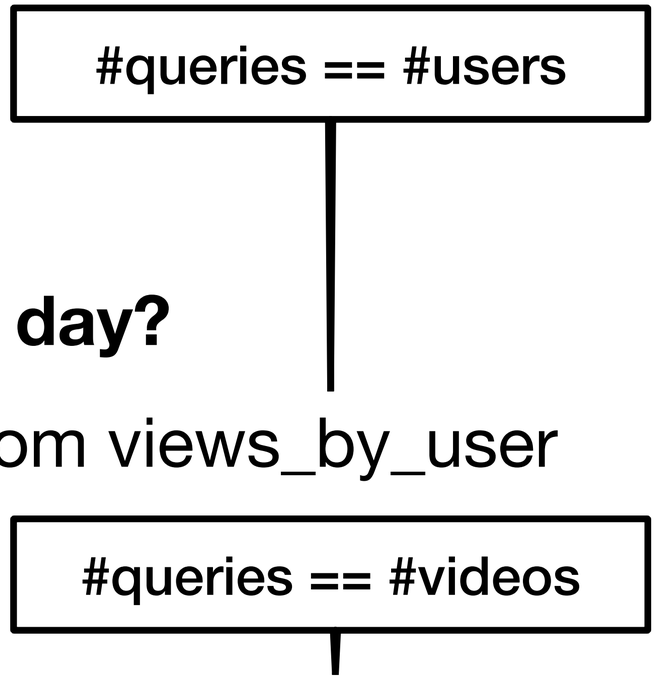
- Create a job (Spark?) that reads all the views from views_by_user and filter the result

- Create a job (Spark?) that reads all the views from views_by_video and filter the result

- Denormalize to another table

What is the difference between the options (time / IO)?

Example continue



How can we return all the views for a specific day?

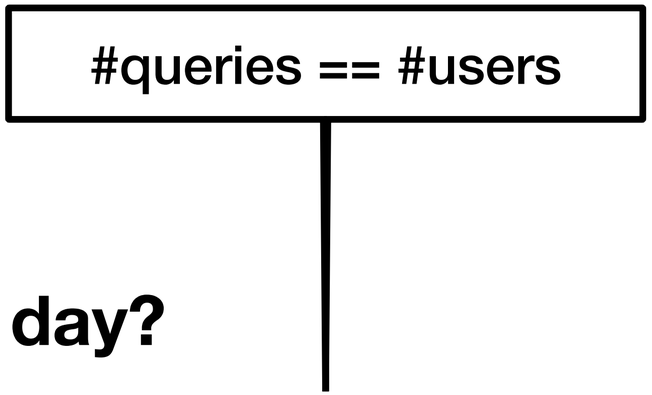
- Create a job (Spark?) that reads all the views from views_by_user and filter the result
- Create a job (Spark?) that reads all the views from views_by_video and filter the result
- Denormalize to another table

What is the difference between the options (time / IO)?

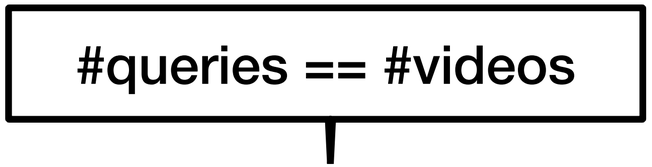
Example continue

How can we return all the views for a specific day?

- Create a job (Spark?) that reads all the views from views_by_user and filter the result



- Create a job (Spark?) that reads all the views from views_by_video and filter the result



- Denormalize to another table

What is the difference between the options (time / IO)?

Example continue

How can we return all the views for a specific day?

- Create a job (Spark?) that reads all the views from views_by_user and filter the result

#queries == #users

- Create a job (Spark?) that reads all the views from views_by_video and filter the result

#queries == #videos

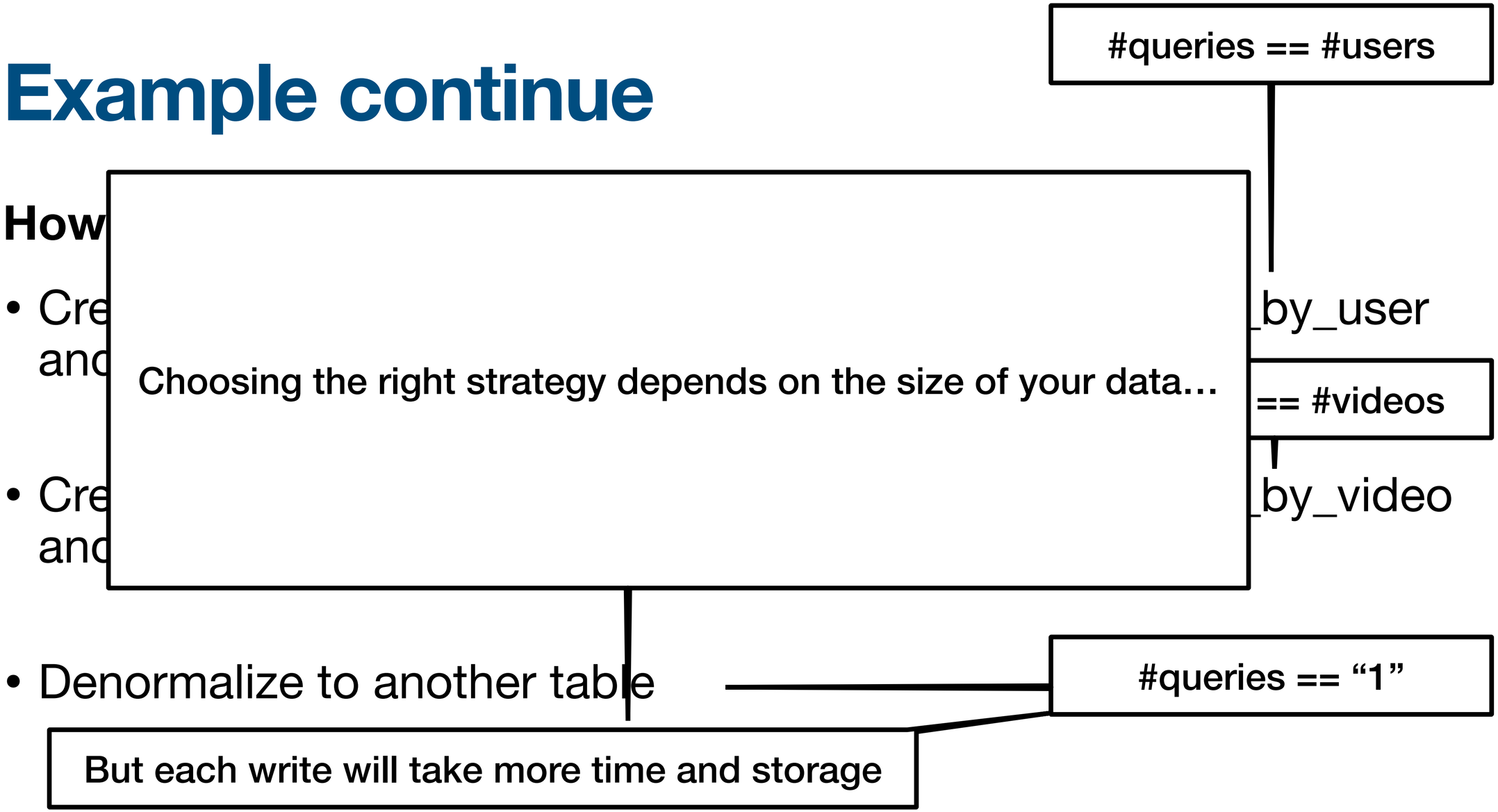
- Denormalize to another table

#queries == “1”

But each write will take more time and storage

What is the difference between the options (time / IO)?

Example continue



What is the difference between the options (time / IO)?

3rd denormalization

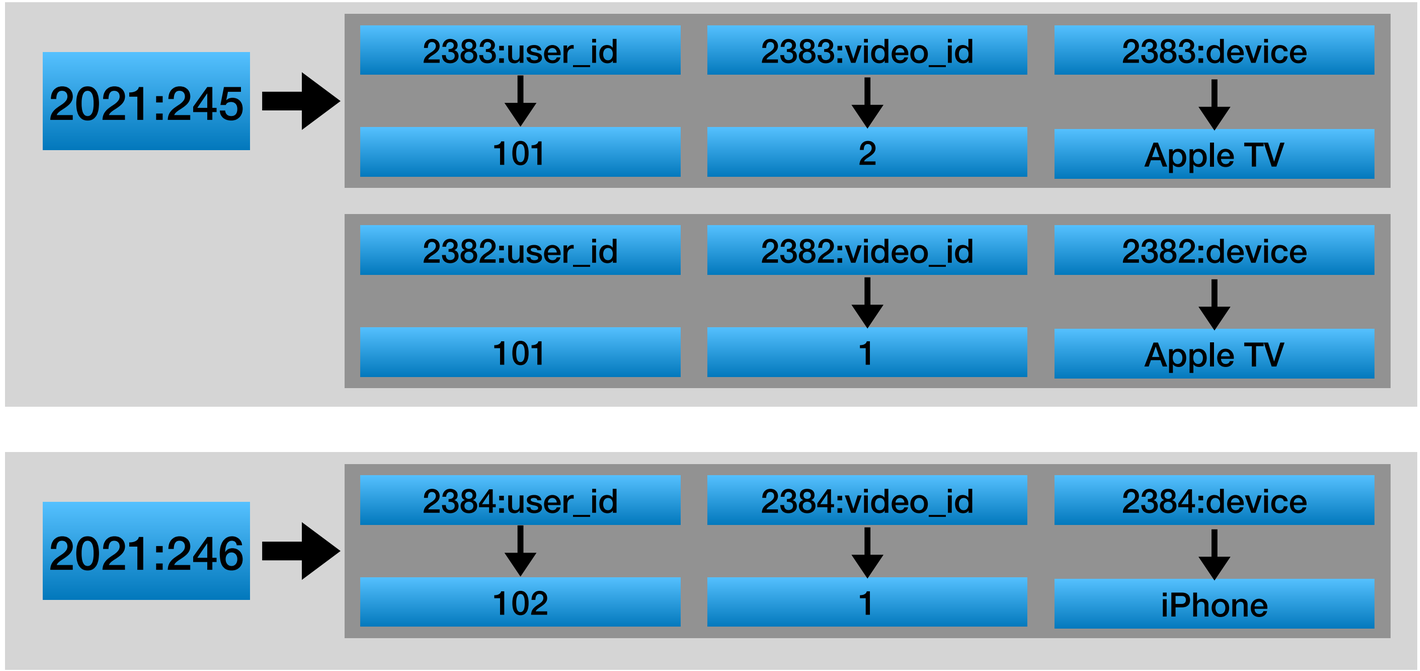
```
CREATE TABLE views_by_user (
    user_id         BIGINT,
    view_id         TIMEUUID,
    video_id        BIGINT,
    device          TEXT,
    PRIMARY KEY ((user_id), view_id)
) WITH CLUSTERING ORDER BY
  (view_id DESC);
```

```
CREATE TABLE views_by_video (
    video_id        BIGINT,
    view_id         TIMEUUID,
    user_id         BIGINT,
    device          TEXT,
    PRIMARY KEY ((video_id), view_id)
) WITH CLUSTERING ORDER BY
  (view_id DESC);
```

```
CREATE TABLE views_by_day (
    year            INT,
    day             INT,
    view_id         TIMEUUID,
    user_id         BIGINT,
    video_id        BIGINT,
    device          TEXT,
    PRIMARY KEY ((year, day), view_id)
) WITH CLUSTERING ORDER BY
  (view_id DESC);
```

3rd denormalization

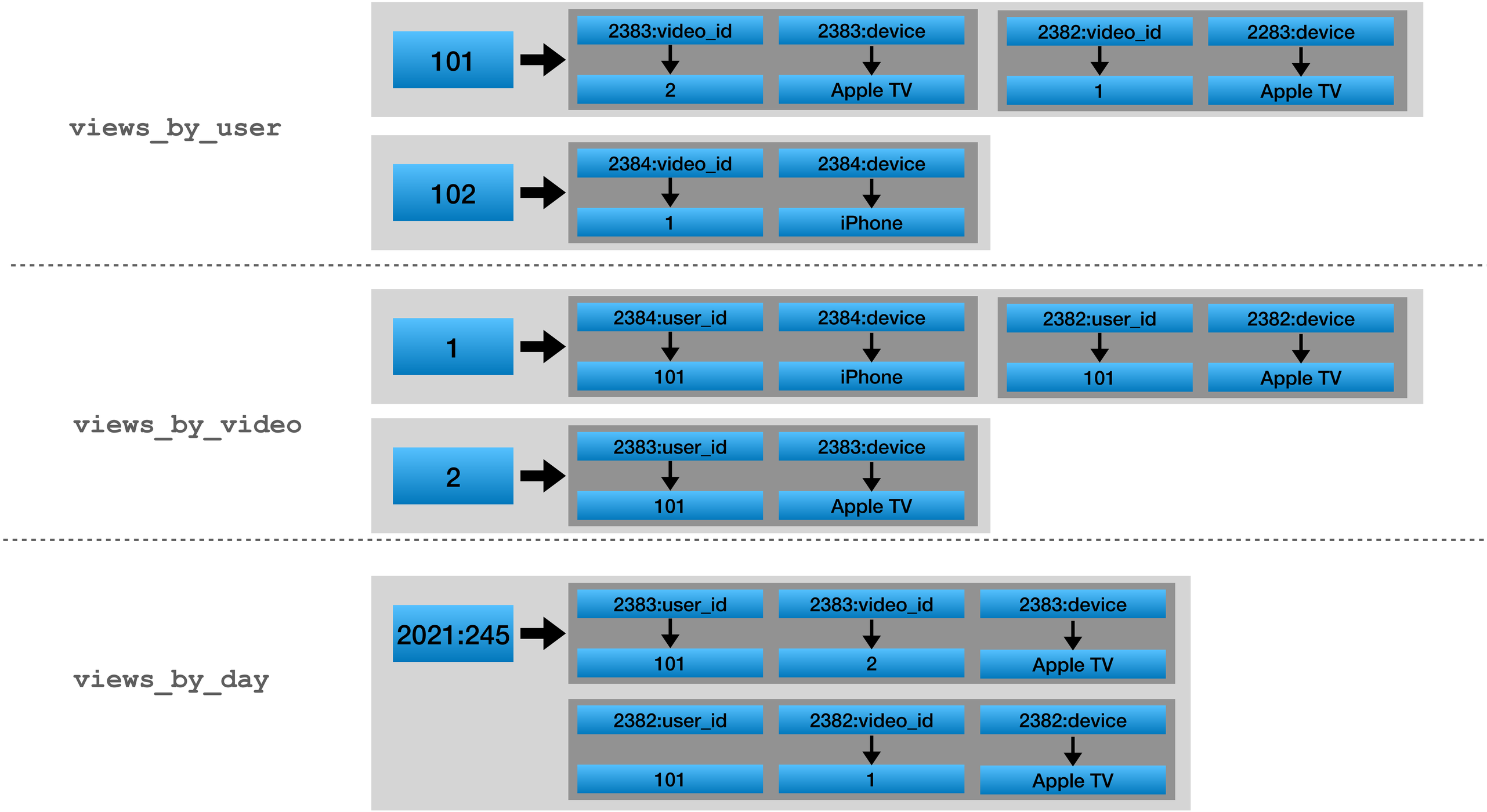
`views_by_day`



assuming:
- views 2382 and 2383 are on day 245 (2021)
- view 2384 is on day 246 (2021)

views

view_id	user_id	video_id	device	...
2382	101	1	Apple TV	
2383	101	2	Apple TV	
2384	102	1	iPhone	


views_by_user
101
2383:video_id
2
2383:device
Apple TV
2382:video_id
1
2283:device
Apple TV
102
2384:video_id
1
2384:device
iPhone
views_by_video
1
2384:user_id
101
2384:device
iPhone
2382:user_id
101
2382:device
Apple TV
2
2383:user_id
101
2383:device
Apple TV
views_by_day
2021:245
2383:user_id
101
2383:video_id
2
2383:device
Apple TV
2382:user_id
101
2382:video_id
1
2382:device
Apple TV

Interesting questions

What happens if

- Barkuni releases new video?
- Taylor Swift releases new song?
- Elon Musk wakes up?
- We have +100m views per day?

* image from YouTube