

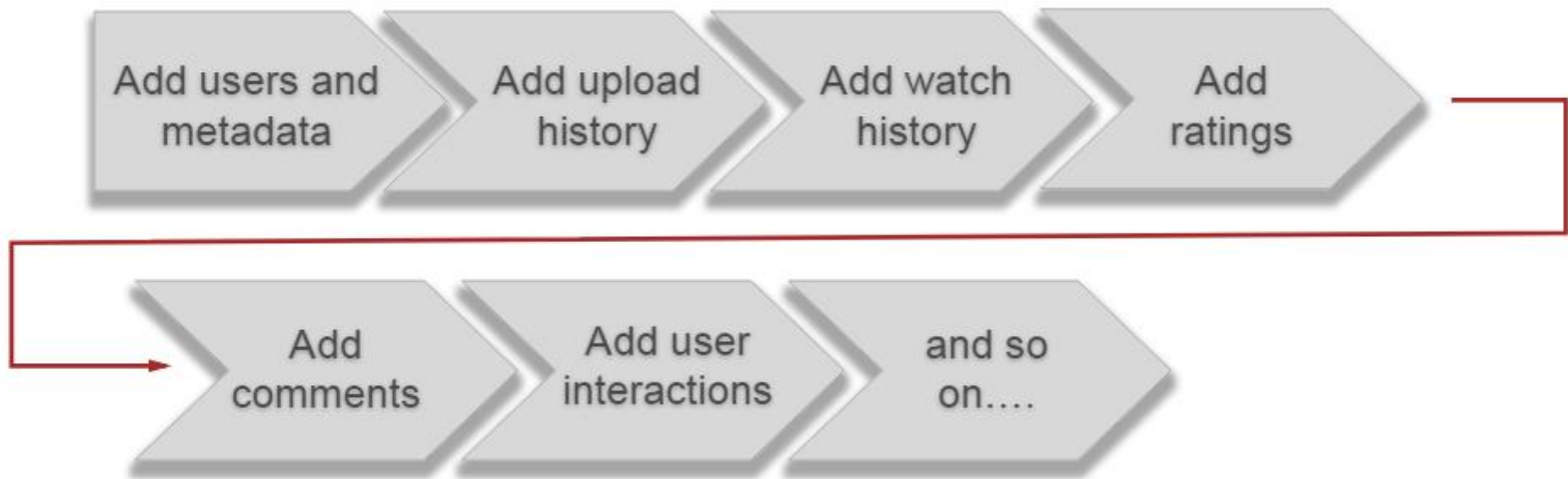
Apache Cassandra

A COLUMN-BASED NOSQL DATABASE

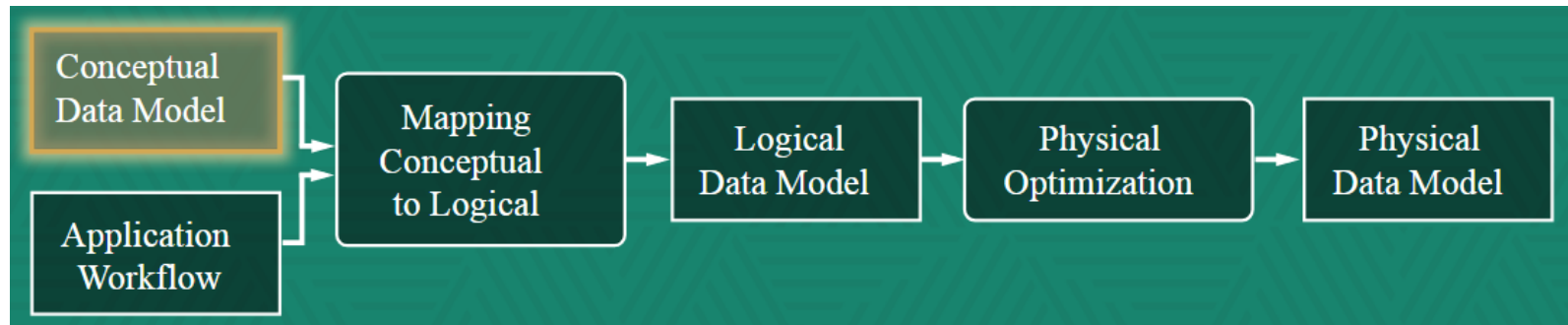
Progettazione

Abbiamo visto come modellare alcune tabelle per rispondere a specifiche query...

... ma come progettare la modellazione di un intero database?



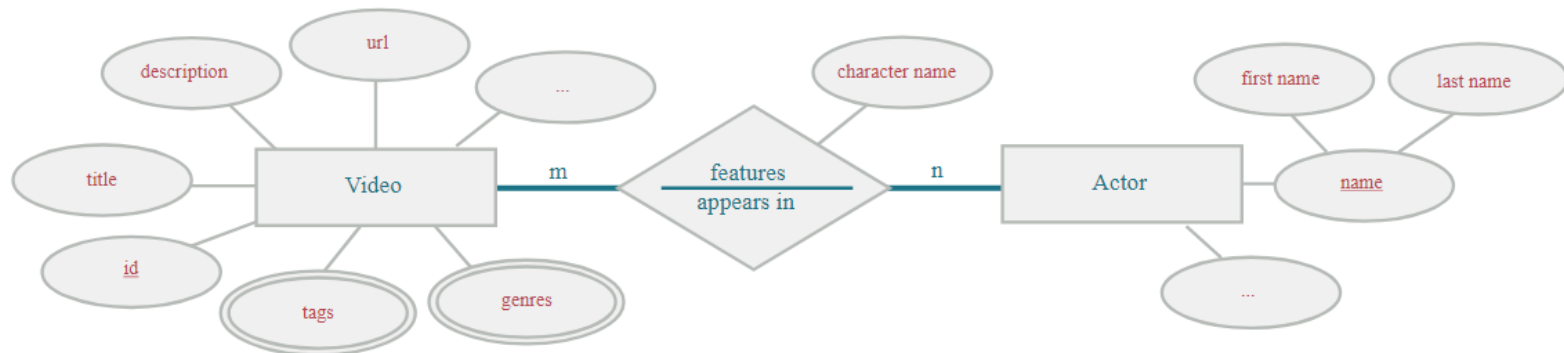
Metodologia



Modellazione concettuale

Modello quasi-ER

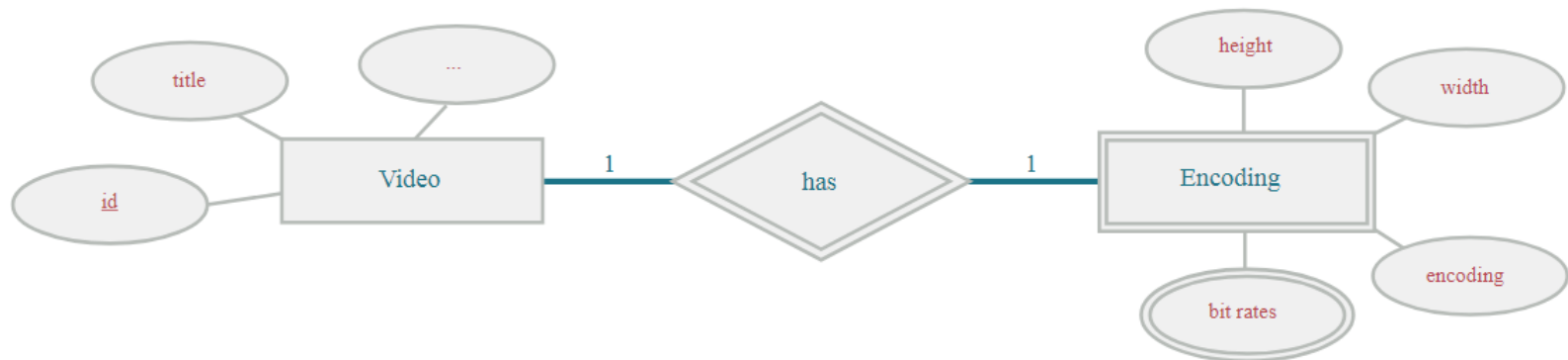
- Le cardinalità sugli archi sono cardinalità massime
- Gli attributi sottolineati sono chiavi primarie
- Gli attributi con un doppio ovale rappresentano collezioni



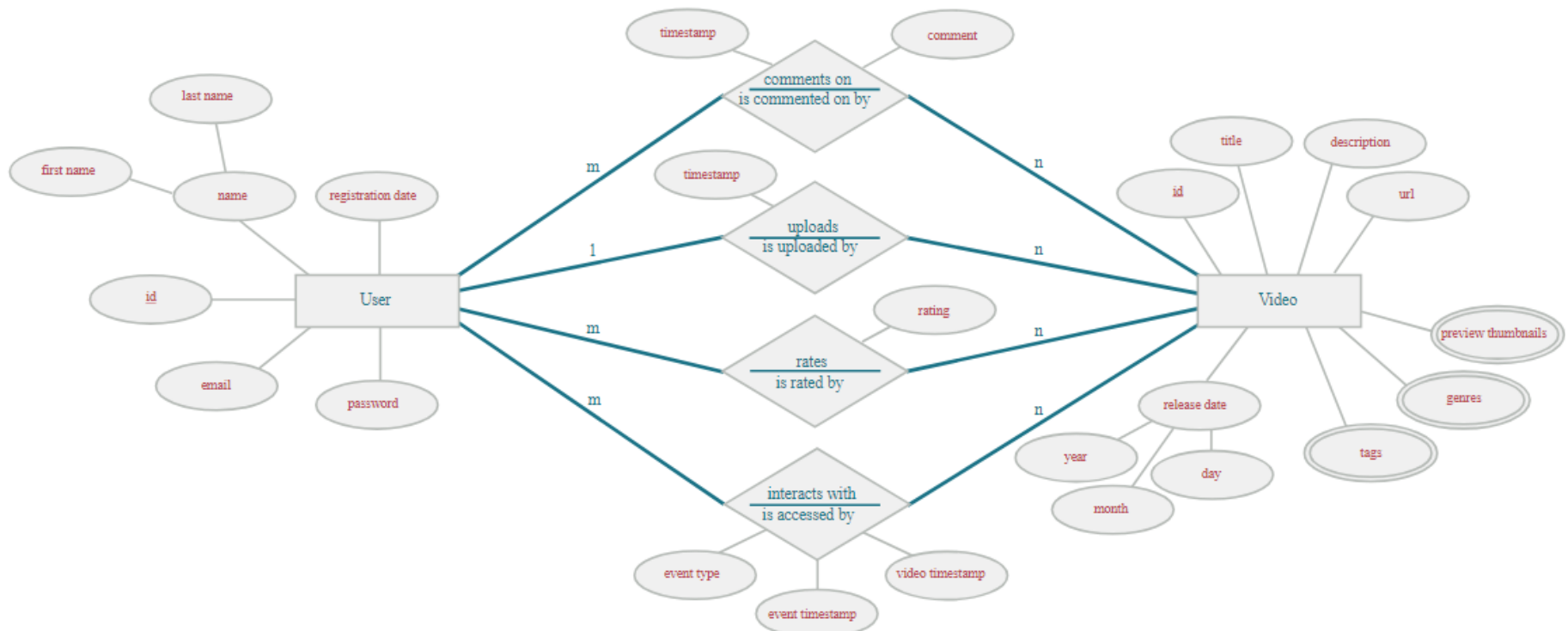
Modello quasi-ER

Entità e relazioni con doppio rettangolo e doppio rombo rappresentando relazioni *part-of*

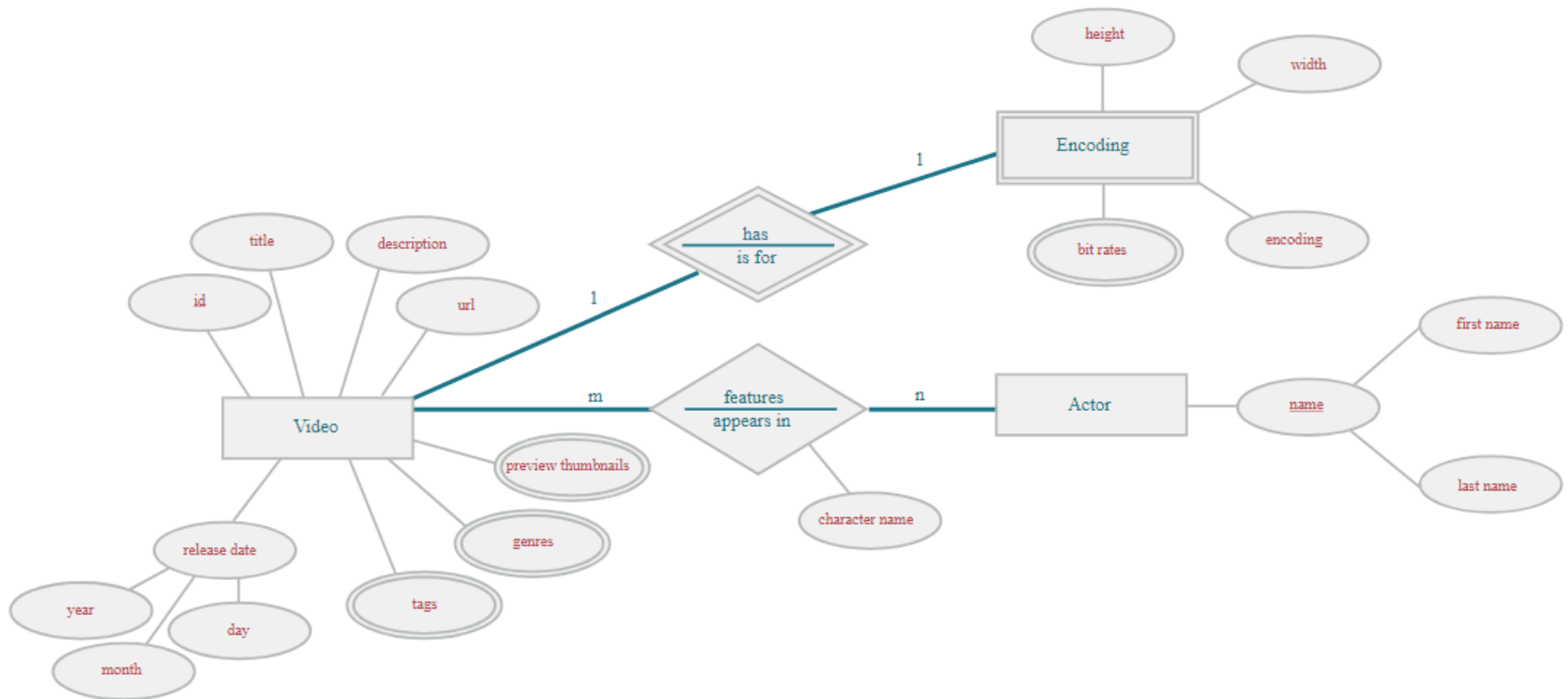
- Un'istanza di encoding non può esistere senza un'istanza di video ad essa associata



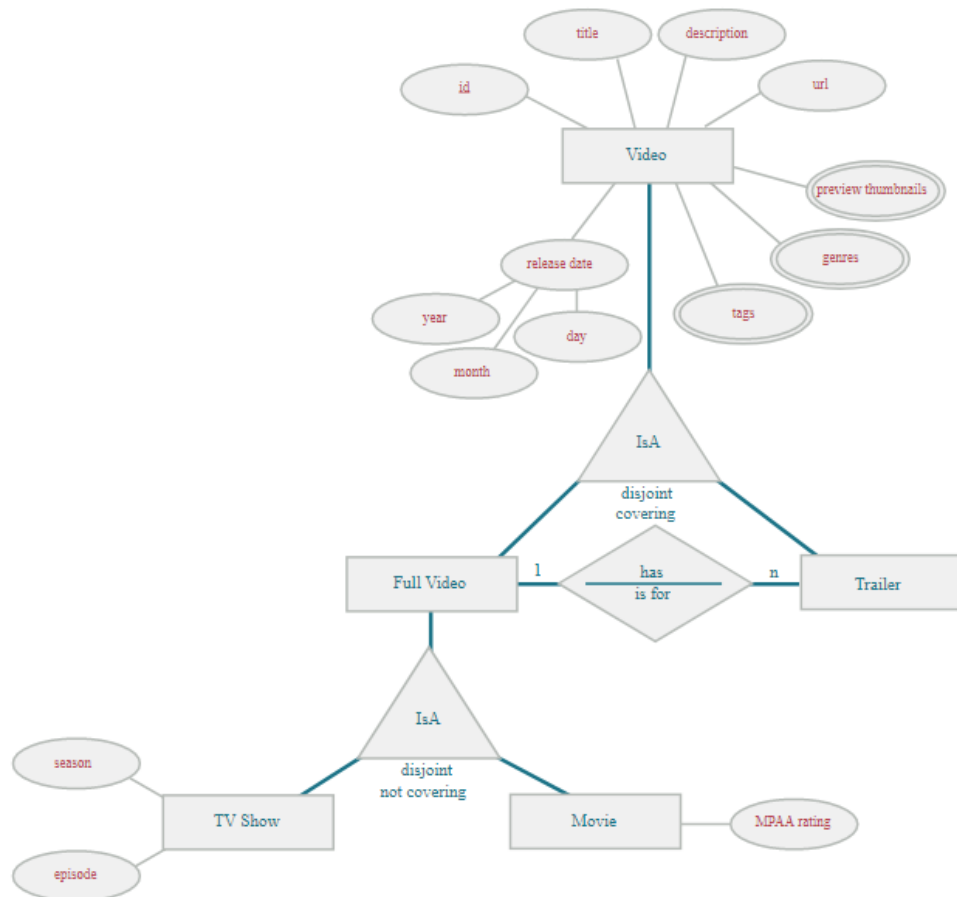
Video e utenti



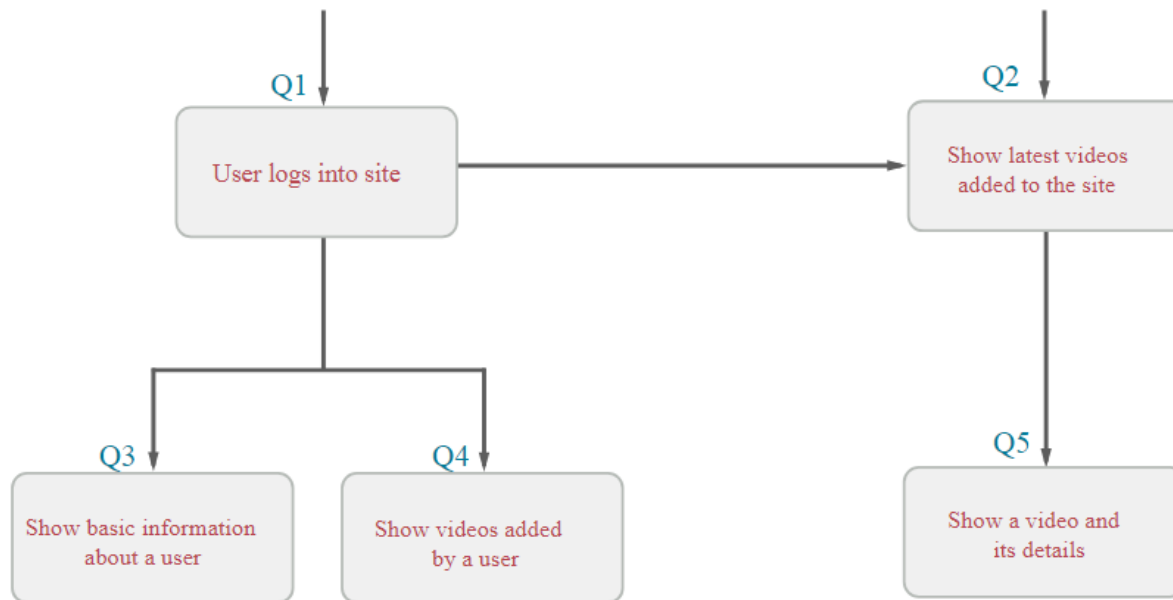
Video, codifiche e attori



Classificazione dei video



Application workflow



ACCESS PATTERNS

Q1: Find a user with a **specified email**

Q2: Find most recently uploaded **videos**

Q3: Find a **user with a specified id**

Q4: Find videos uploaded by a **user with a known id** (show most recently uploaded videos first)

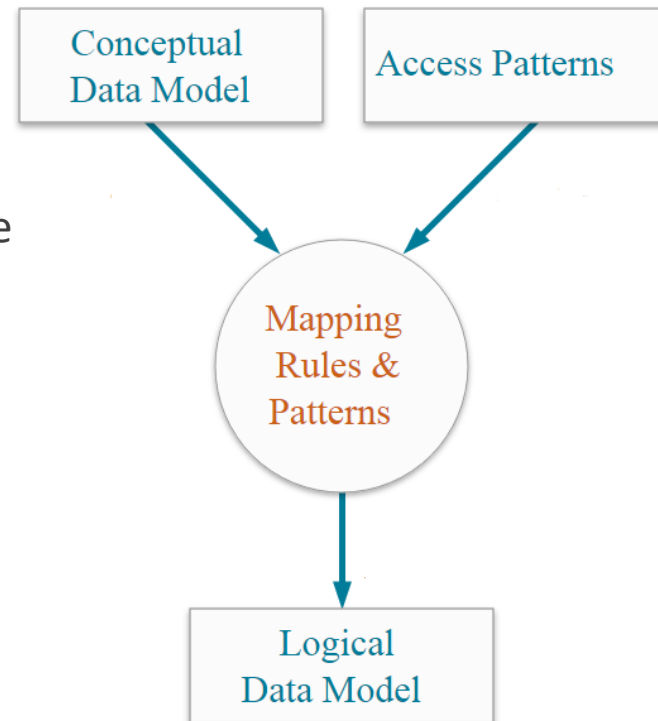
Q5: Find a video with a **specified video id**

Modellazione logica

Approccio query-driven

Dati il modello concettuale ed il workflow dell'applicazione, il modello logico può essere ricavato seguendo un approccio query-driven

- Regole di mapping garantiscono la correttezza del modello logico
- Ad ogni query la sua tabella
- Le tabelle vengono progettate per garantire una corretta esecuzione delle query
- Le tabelle restituiscono i dati nell'ordine corretto



Modello logico

Dati il modello logico ed il workflow dell'applicazione, il modello logico può essere ricavato applicando **in ordine** le seguenti 5 regole che riguardano, rispettivamente:

1. Entità e relazioni
2. Attributi per ricerche esatte
3. Attributi per ricerche inesatte
4. Attributi per ordinamento
5. Attributi chiave

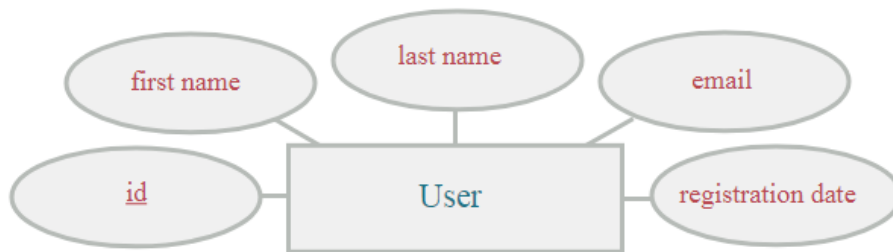
Regola 1

Entità e relazioni

Traduzione di entità

- Una tabella per ogni entità
- Una colonna per ogni attributo

Conceptual Model



Logical Model

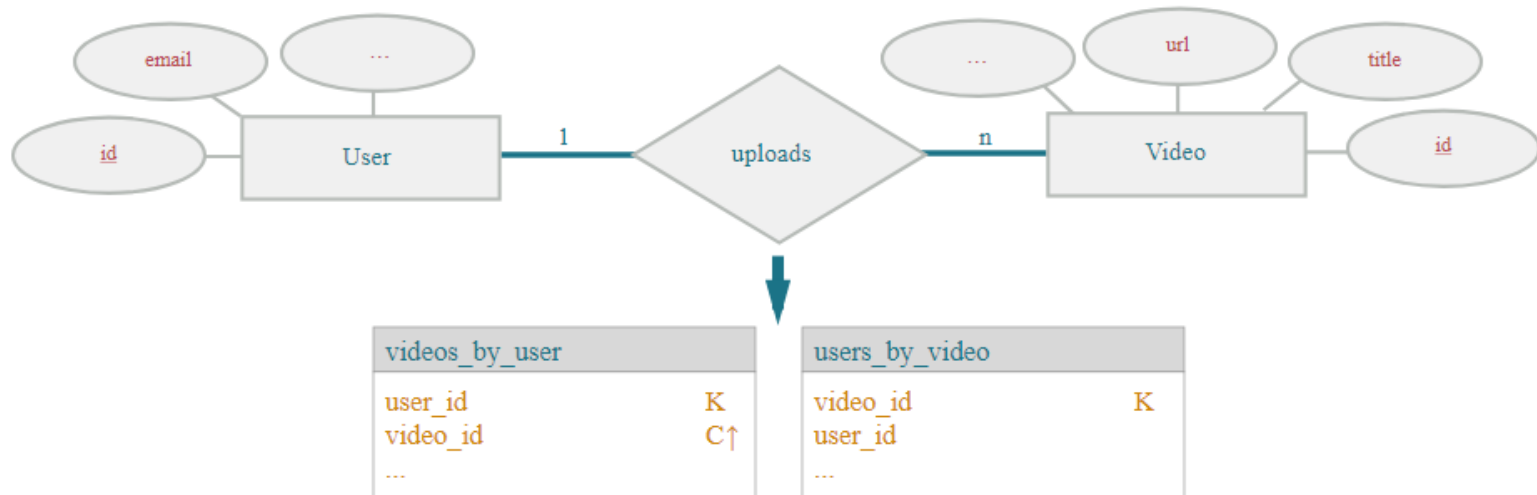
users	
user_id	K
email	
first_name	
last_name	
registration_date	

Regola 1

Entità e relazioni

Traduzione di relazioni

- Una tabella per ogni relazione
- Attenzione: possono esistere soluzioni diverse
- La scelta su quale implementare dipende dalla cardinalità della relazione e dal carico di lavoro



Regola 2

Attributi per ricerche esatte

Attributi su cui effettuare ricerche esatte

- La chiave primaria è un insieme ordinato di colonne, costituito da chiave di partizionamento e colonne di raggruppamento
- La chiave di partizionamento è formato da una (o più) delle prime colonne dichiarate nella chiave primaria
- Le query devono sempre includere le colonne della chiave di partizionamento

PRIMARY KEY ((partition-key1, partition-key2, ...), clustering-column1, clustering-column2, ...)

Equality search

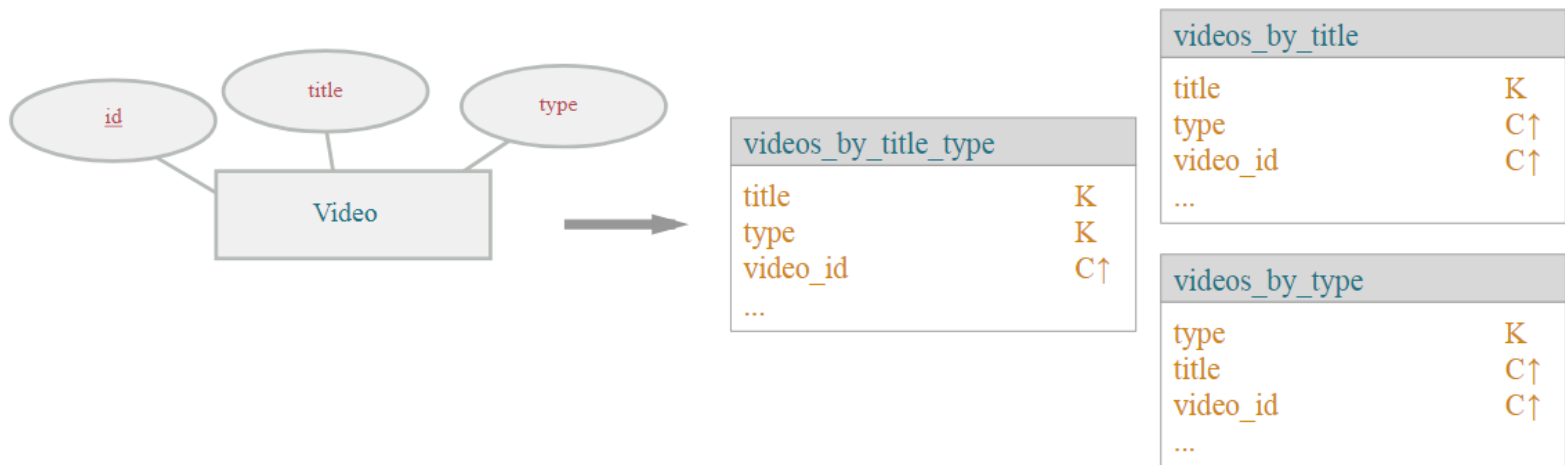
The diagram illustrates the concept of 'Equality search' in the context of a primary key. It shows a primary key definition: PRIMARY KEY ((partition-key1, partition-key2, ...), clustering-column1, clustering-column2, ...). Arrows point from the text 'Equality search' to the partition key part (partition-key1, partition-key2, ...) and the clustering columns part (clustering-column1, clustering-column2, ...).

Regola 2

Attributi per ricerche esatte

Gli attributi su cui effettuare ricerche esatte devono essere le prime colonne della chiave primaria

- Ricerca su *title* e *type*
- 3 possibili soluzioni

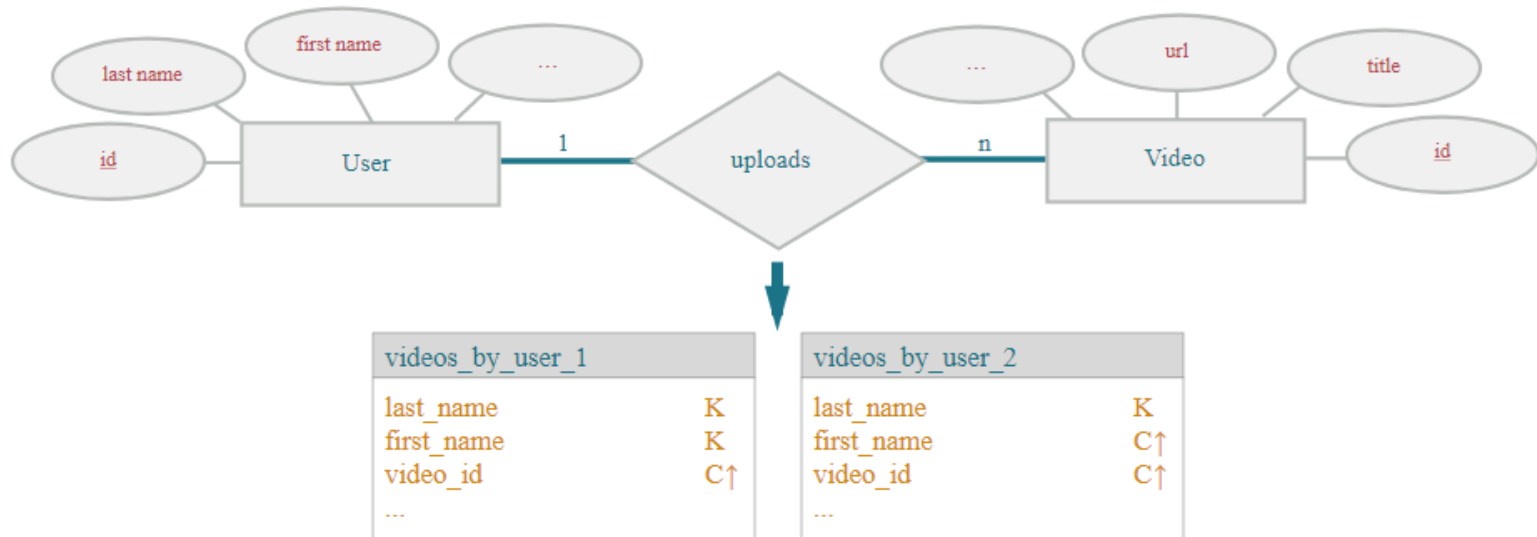


Regola 2

Attributi per ricerche esatte

Gli attributi su cui effettuare ricerche esatte devono essere le prime colonne della chiave primaria

- Ricerca su *last_name* e *first_name*
- 2 possibili soluzioni

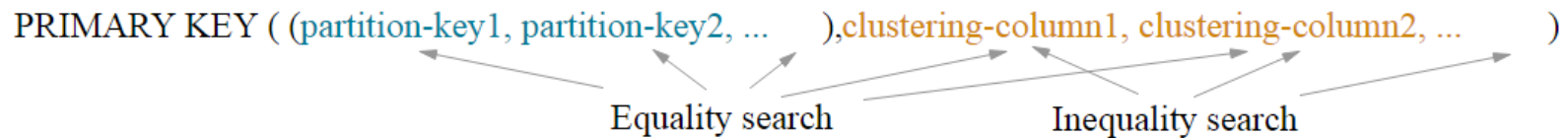


Regola 3

Attributi per ricerche inesatte

Attributi su cui effettuare ricerche inesatte

- Sulla chiave di partizionamento è possibile fare solo ricerche esatte
- Gli attributi per ricerche inesatte vanno dichiarati come colonne di clustering, DOPO gli attributi su cui fare ricerche esatte



Regola 3

Attributi per ricerche inesatte

Attributi su cui effettuare ricerche inesatte

- Ricerca su *last_name* = ? e *registration_date* > ?

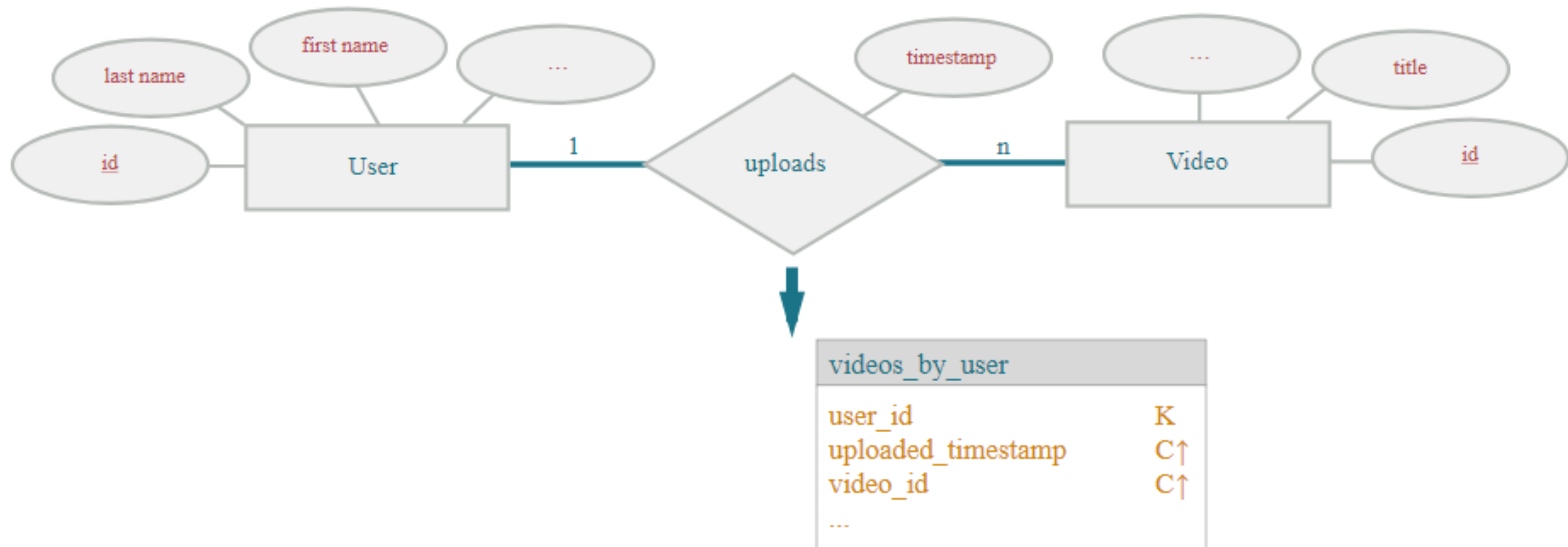


Regola 3

Attributi per ricerche inesatte

Attributi su cui effettuare ricerche inesatte

- Ricerca su *user_id* = ? e *uploaded_timestamp* > ?



Regola 4

Attributi per ordinamenti

Attributi su cui effettuare ordinamenti

- In ogni tabella, le partizioni sono ordinate sulle colonne di raggruppamento, nell'ordine in cui sono state dichiarate
- L'ordinamento va specificato in fase di creazione
- Non esiste la clausola di ORDER BY in CQL

```
PRIMARY KEY ( (partition-key1, partition-key2, ... ), clustering-column1, clustering-column2, ... )
```

↓

```
ORDER BY clustering-column1 ASC, clustering-column2 ASC, ...
```

Regola 4

Attributi per ordinamenti

Attributi su cui effettuare ordinamenti

- Ricerca su *last_name* = ? e *registration_date* > ?
- Ordinamento per *registration_date* crescente

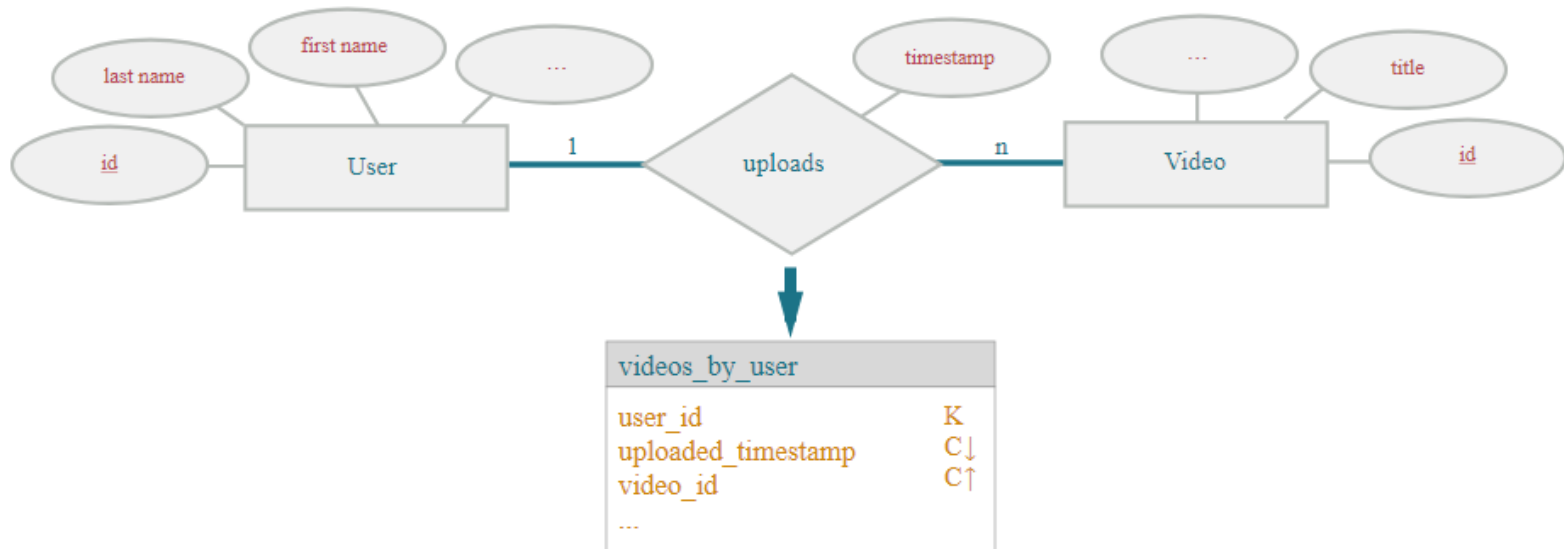


Regola 4

Attributi per ordinamenti

Attributi su cui effettuare ordinamenti

- Ricerca su *user_id* = ? e *uploaded_timestamp* > ?
- Ordinamento per *uploaded_timestamp* decrescente e *video_id* crescente

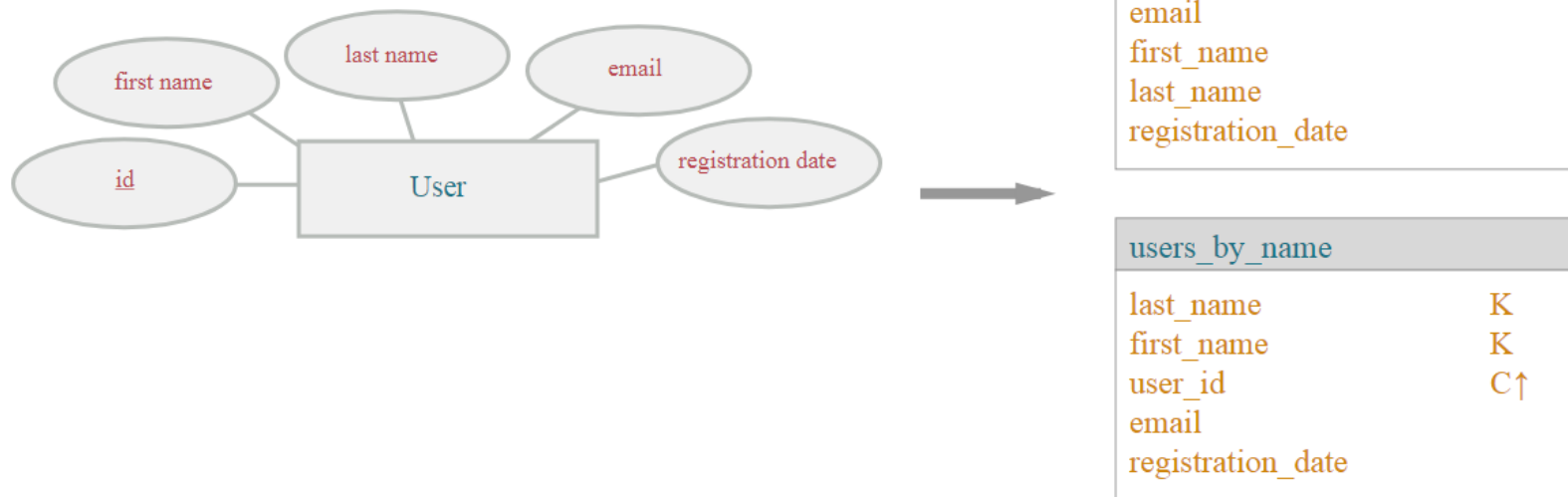


Regola 5

Attributi chiave

Gli attributi chiave devono essere dichiarati nella chiave primaria

- Importante per mantenere la granularità giusta per la tabella

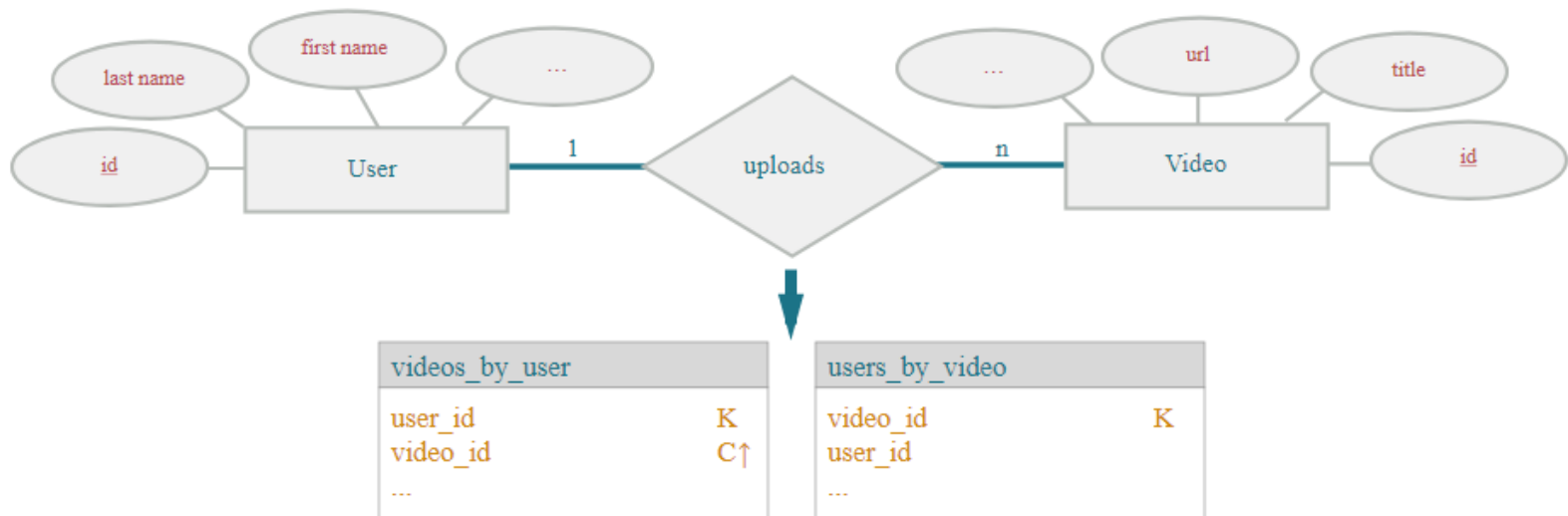


Regola 5

Attributi chiave

Gli attributi chiave devono essere dichiarati nella chiave primaria

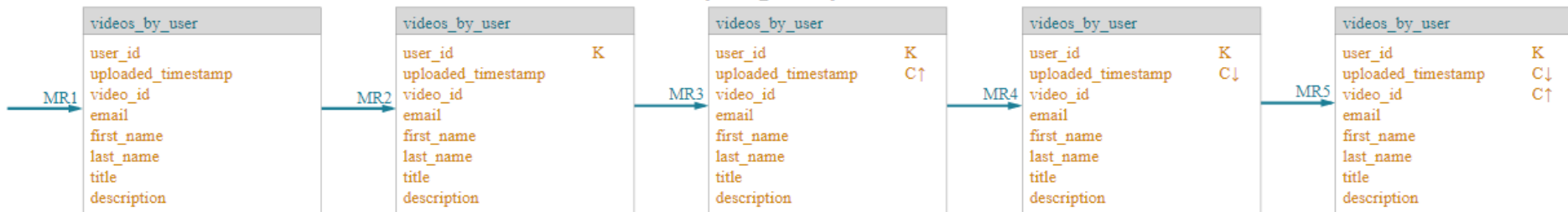
- Importante per mantenere la granularità giusta per la tabella



Un esempio completo

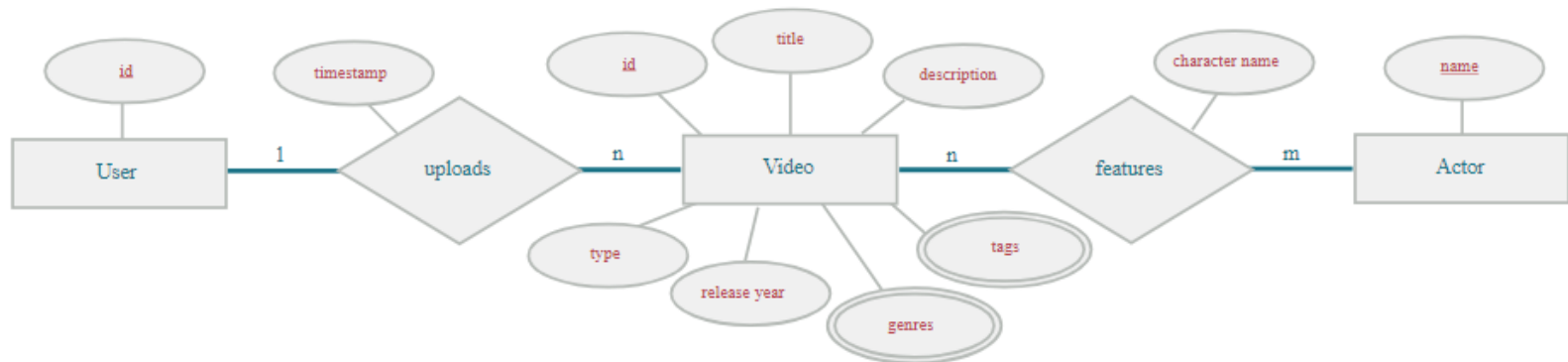


Query: user_id = ? and uploaded_timestamp > ?
ORDER BY uploaded_timestamp DESC



Esercizio 7

Dato il seguente schema concettuale, definire le tabelle che rispondono alle seguenti query



- Q1: Dato un tag, trova tutti i video (mostra prima i più recenti)
- Q2: Trova tutti i video di un dato attore in un dato range di anni (mostra prima i più recenti e ordina anche per titolo)
- Q3. Come Q2, ma filtrando anche per genere

Esercizio 7

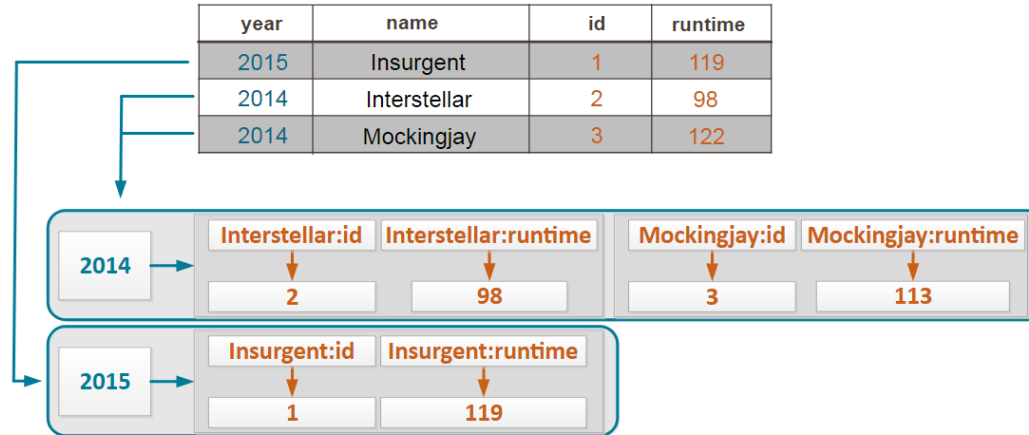
videos_by_tag	
tag	K
timestamp	C↓
video_id	C↑
user_id	
title	
description	
type	
release_year	
{genre}	
{tags}	

videos_by_actor	
actor_name	K
release_year	C↓
title	C↑
video_id	C↑
character_name	C↑
description	
type	
{genre}	
{tags}	

videos_by_genre_actor	
genre	K
actor_name	K
release_year	C↓
title	C↑
video_id	C↑
character_name	C↑
description	
type	
{genre}	
{tags}	

Colonne statiche

La struttura della chiave primaria (partizionamento + raggruppamento) consente di memorizzare più righe nella stessa partizione

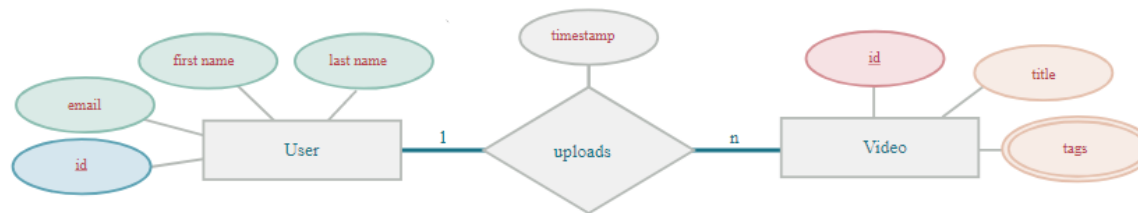


Se una colonna è funzionalmente determinata dalla chiave di partizionamento, è possibile dichiararla come statica

- In questo modo ce ne sarà una per partizione, non per ogni riga

Colonne statiche

Quando usarle



ACCESS PATTERN

WHERE user_id = ?

videos_by_user	
user_id	K
video_id	C↑
email	S
first_name	S
last_name	S
title	
{tags}	
uploaded_timestamp	

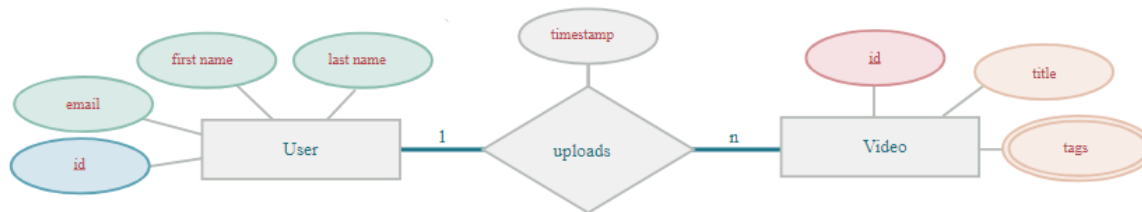
ACCESS PATTERN

WHERE user_id = ? AND uploaded_timestamp > ?

videos_by_user	
user_id	K
uploaded_timestamp	C↑
video_id	C↑
email	S
first_name	S
last_name	S
title	
{tags}	

Colonne statiche

Quando non usarle



ACCESS PATTERN

WHERE first_name = ? AND last_name = ?

videos_by_user	
first_name	K
last_name	K
video_id	C↑
title	
{tags}	
uploaded_timestamp	

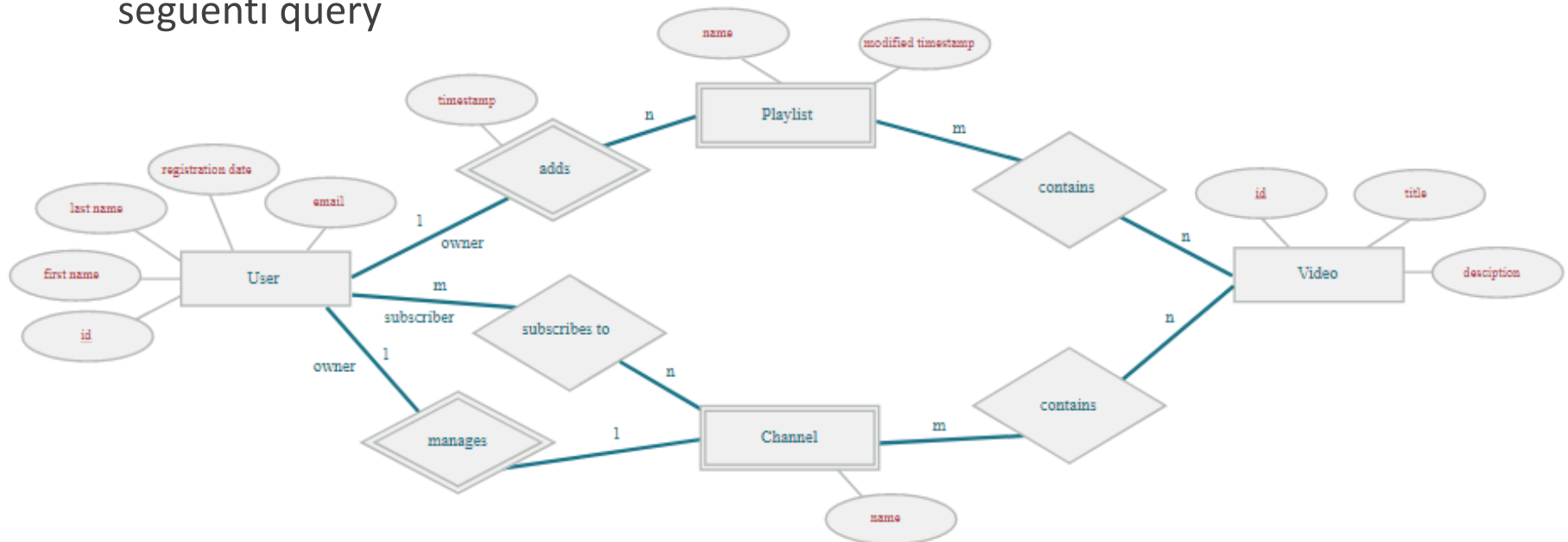
ACCESS PATTERN

WHERE email = ? AND uploaded_timestamp > ?

videos_by_user	
email	K
uploaded_timestamp	C↑
video_id	C↑
title	
{tags}	

Esercizio 8

Dato il seguente schema concettuale, definire le tabelle che rispondono alle seguenti query



- Q1: Trova tutte le playlist dati il proprietario ed un range di date di modifica (mostra prima quelle modificate più di recente)
- Q2: Trova tutti i canali (ordinate alfabeticamente) a cui un dato utente è registrato
- Q3. Trova tutti i video contenuti in uno specifico canale

Esercizio 8

playlist_by_owner	
owner_id	K
modified_timestamp	C↓
name	C↑

channels_by_subscriber	
subscriber_id	K
channel_name	C↑
owner_id	C↑
first_name	S
last_name	S
registration_date	S
email	S

videos_by_channel	
owner_id	K
video_id	C↑
channel_name	S
title	
description	

Modellazione fisica

Le regole di modellazione logica definiscono delle *best practices* per costruire tabelle coerenti coi requisiti dell'applicazione

Tuttavia, rimane del margine per ottimizzare le tabelle dal punto di vista dell'efficienza, considerando:

- Le limitazioni del database
- Le risorse del cluster (numero di nodi, spazio su disco, memoria, ..)
- Eventuali operazioni non supportate direttamente dal DBMS

Dimensione delle partizioni

Uno degli elementi più importanti da controllare

- Una partizione è atomica, indivisibile
- Deve risiedere interamente in un nodo

Limite di Cassandra: massimo 2 miliardi di colonne per partizione

- Best practice: non superare l'ordine delle centinaia di MB per partizione

Ridondanza dei dati

A causa delle regole di modellazione (niente normalizzazione, una tabella per query), il database conterrà più copie degli stessi dati

- Da non confondere col meccanismo di replication
- Esempio: dataset da 10TB, ridondanza 10x, replicazione 5x; totale 500TB

Evitare fattori di ridondanza non costanti (e.g., n , n^2 , 2^n , ecc.)

- Quanti tag può avere un video?
- In caso di ridondanza non costante, rivedere il caso d'uso o imporre dei limiti

Non dimenticare il problema della consistenza

videos_by_actor_tag

actor	K
tag	C↑
video_id	C↑
title	
type	
{tags}	
<preview_thumbnails>	

Consistenza dei dati

La consistenza dei dati va gestita a livello di applicazione

- Una modifica può comportare l'esecuzione di tante INSERT/UPDATE/DELETE

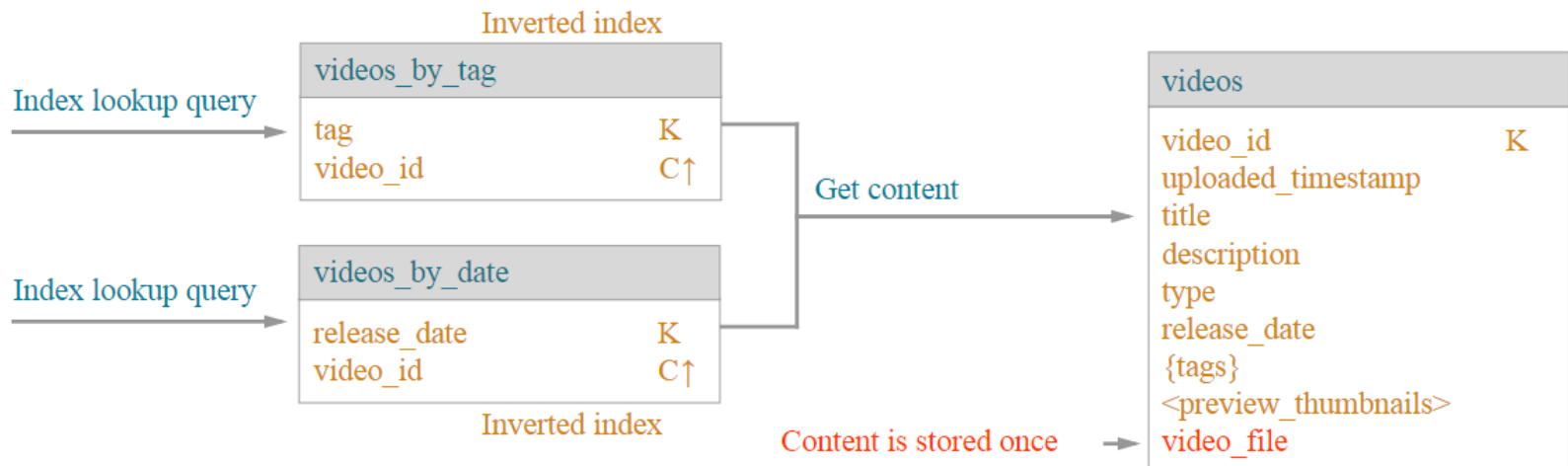
Cassandra fornisce un meccanismo di esecuzione batch

```
BEGIN BATCH
  UPDATE videos SET title = 'Jaws' WHERE video_id = 1;
  INSERT INTO videos_by_title (title, video_id, ...) VALUES ('Jaws', 1);
  DELETE FROM videos_by_title WHERE title = 'Jaw' AND video_id = 1;
APPLY BATCH;
```

- Garantisce l'atomicità dell'operazione
- Non prevede rollback
- Non garantisce isolation rispetto a letture e scritture concorrenti
- Non ottimizza le performance

Join

- CQL non supporta i join
- Materializzare ogni query in una tabella comporta costi legati alla ridondanza dei dati
- Una soluzione è quella di ristrutturare le tabelle delle query come dei semplici indici verso la tabella principale
- Il join va però gestito a livello di applicazione



Aggregazioni

Cassandra non supporta GROUP BY

- Unico raggruppamento supportato: COUNT(*)

Come fare aggregazioni?

- Creare una tabella che contiene dati aggregati e aggiornarla continuamente
- Sfruttare il tipo di dato counter
 - Anche se non è accurato al 100%
- Gestire l'aggregazione a livello di applicazione
- Sfruttare altri strumenti che permettono di effettuare aggregazioni e che si interfacciano con Cassandra (e.g., Apache Spark)

ratings_by_video	
video_id	K
num_ratings	++
sum_ratings	++

Transazioni

Cassandra non supporta transazioni ACID...

- Atomicity, Consistency, Isolation, Durability

... ma fornisce costrutti che supportano **parzialmente** queste proprietà:

- Contatori
 - Leggono il valore corrente prima di incrementarlo/decrementarlo
- Esecuzioni batch
 - Garantiscono l'atomicità di un insieme di istruzioni
- INSERT INTO ... IF NOT EXISTS;
 - Evita che venga effettuato un update nel caso il dato esista già
- UPDATE ... IF ...;
 - Simile al counter: verifica un valore prima di fare l'update

```
UPDATE users
SET reset_token = null,
    password = 'trustno1'
WHERE user_id = 'tlberglund'
IF reset_token = 12345678;
```

Esercizio finale

- Aprire lo script `/root/labwork/exercise-16/killrvideo.cql`
- Nelle prime 4 tabelle, inserire il tipo di dato corretto al posto di "CQL TYPE"
- Eseguire lo script

```
SOURCE '/root/labwork/exercise-16/killrvideo.cql';
```

- Popolare le tabelle

```
USE killr_video;
```

```
COPY videos FROM '/root/labwork/exercise-16/videos.csv' WITH HEADER=true;
```

```
COPY latest_videos FROM '/root/labwork/exercise-16/latest_videos.csv' WITH HEADER=true;
```

```
COPY trailers_by_video FROM '/root/labwork/exercise-16/trailers_by_video.csv' WITH HEADER=true;
```

```
COPY actors_by_video FROM '/root/labwork/exercise-16/actors_by_video.csv' WITH HEADER=true;
```

Esercizio finale

- Query: visualizzare i 50 video più recenti dalla tabella latest_videos
 - Cercare l'ID del film "Gone Girl" ("L'amore bugiardo")
- Query: visualizzare i dati del film dalla tabella videos
 - Quando è stato rilasciato il film? A quali generi appartiene?
- Query: visualizzare gli attori coinvolti nel film e i personaggi da loro interpretati dalla tabella actors_by_video
 - Quale attore ha interpretato il personaggio Desi Collings?
- Query: cercare il trailer del film dalla tabella trailers_by_video
 - Il trailer_id è un ID che riconduce alla tabella videos
- Query: visualizzare i dati del trailer dalla tabella videos

Esercizio finale

```
SELECT * FROM latest_videos LIMIT 50;
```

```
SELECT * FROM videos  
WHERE video_id = 8a657435-0ef2-11e5-91b1-8438355b7e3a;
```

```
SELECT * FROM actors_by_video  
WHERE video_id = 8a657435-0ef2-11e5-91b1-8438355b7e3a;
```

```
SELECT * FROM trailers_by_video  
WHERE video_id = 8a657435-0ef2-11e5-91b1-8438355b7e3a;
```

```
SELECT * FROM videos  
WHERE video_id = 8a65751c-0ef2-11e5-9cac-8438355b7e3a;
```