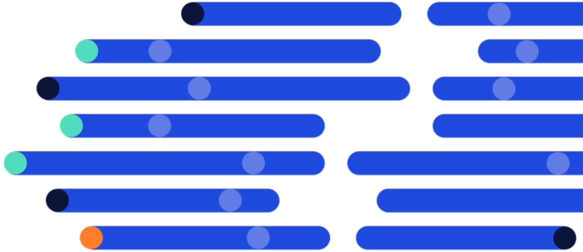
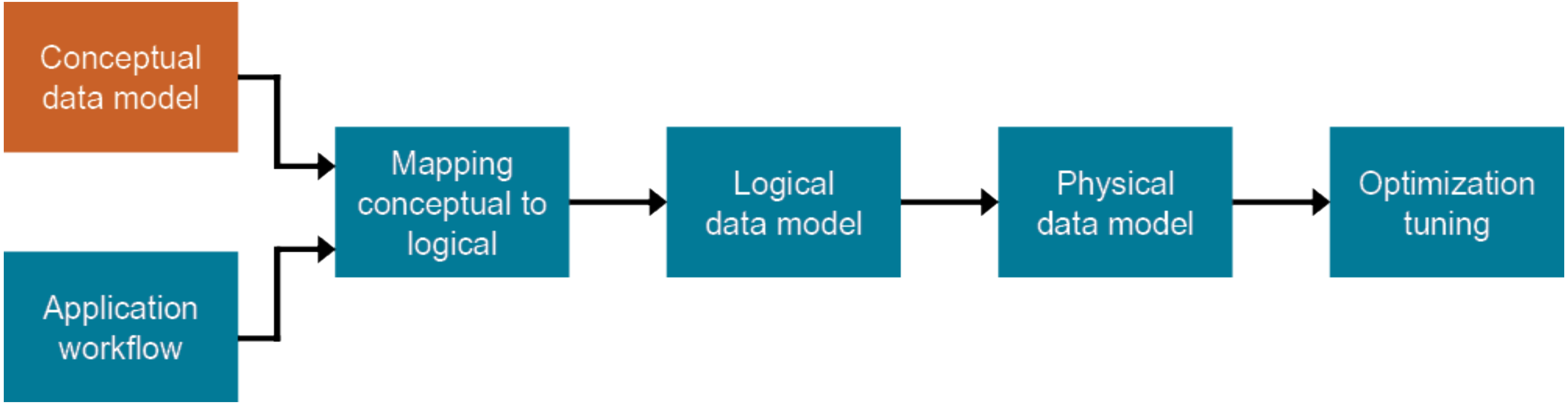
DATASTAX®

Apache Cassandra™ Fasttrack



Data Modeling

Data Modeling Overview



Conceptual data model

Application workflow

Mapping conceptual to logical

Logical data model

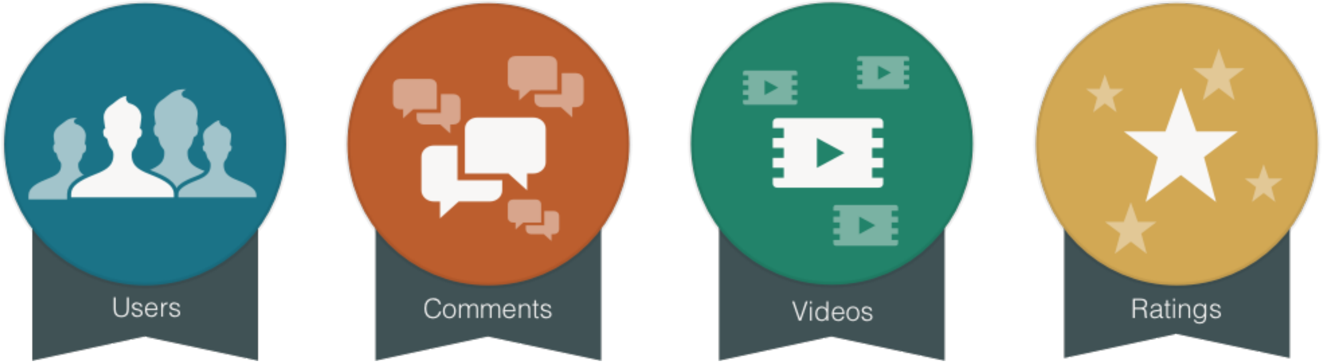
Physical data model

Optimization tuning

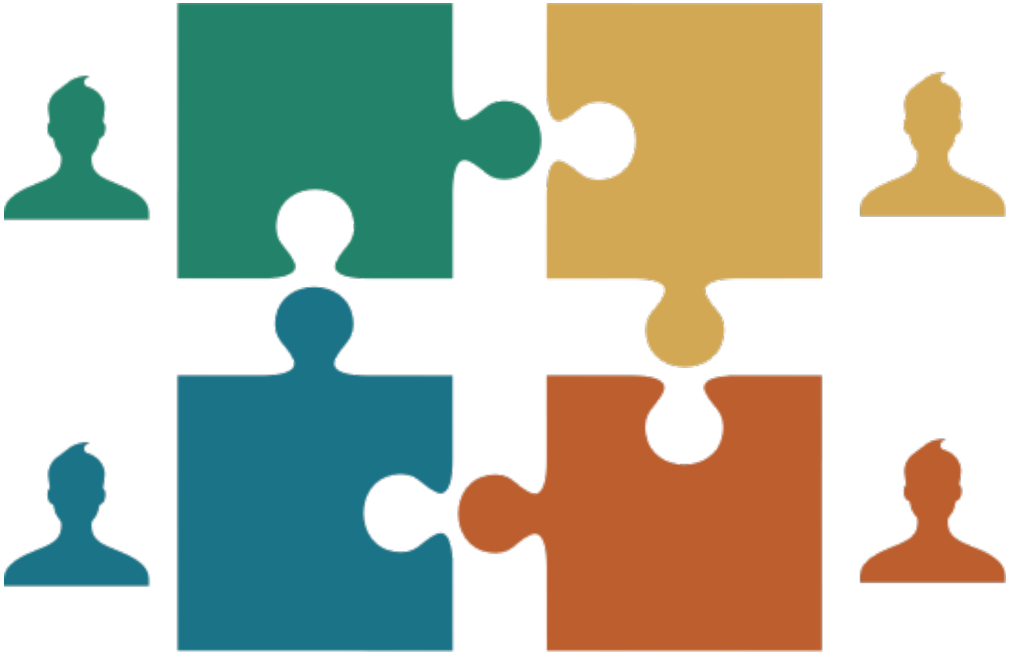
Conceptual Data Model

Purpose of Conceptual Model

- Understand your data
- Essential objects
- Constraints

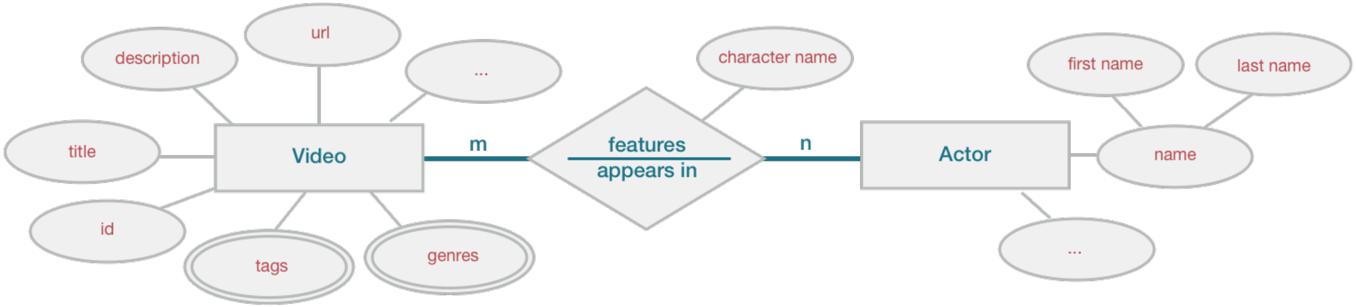


Collaboration = Key



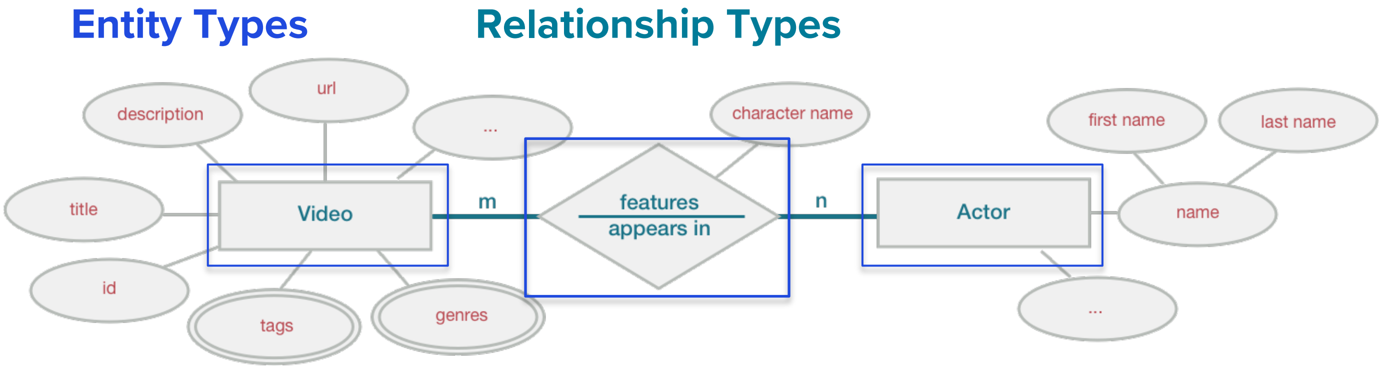
So What Does a Conceptual Model Look Like?

- Abstract view of your domain
- Technology independent
- Not specific to any database system



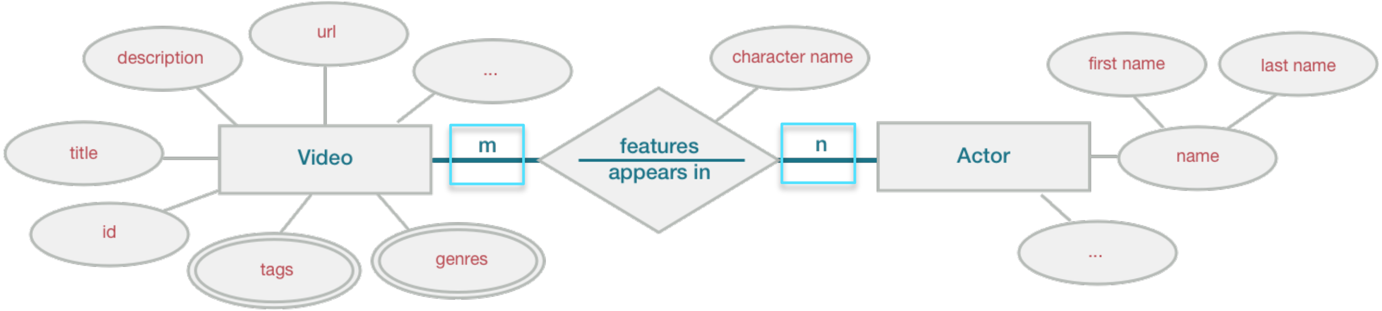
Entity-Relationship (ER) Model

- Entity Types - Relationship Types - Attribute Types



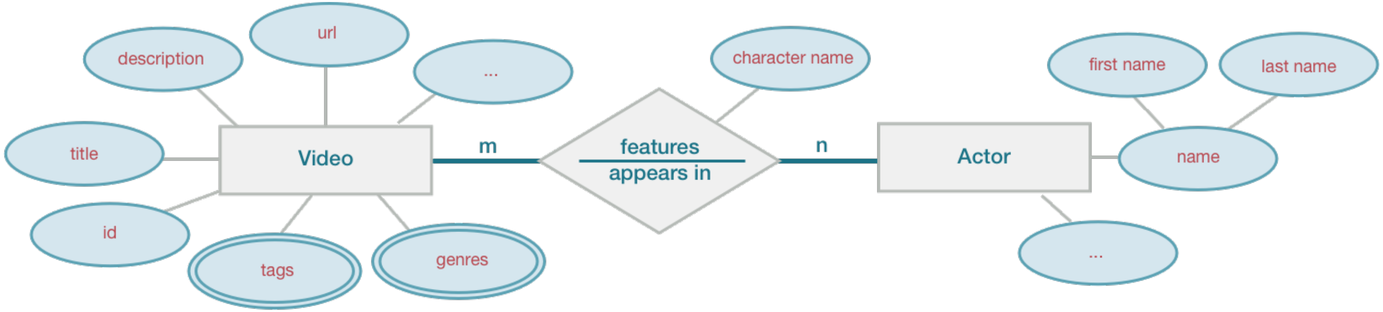
Cardinality

- Number of times an entity can/must participate in the relationship
- Other possibilities:
  - 1-n
  - 1-1



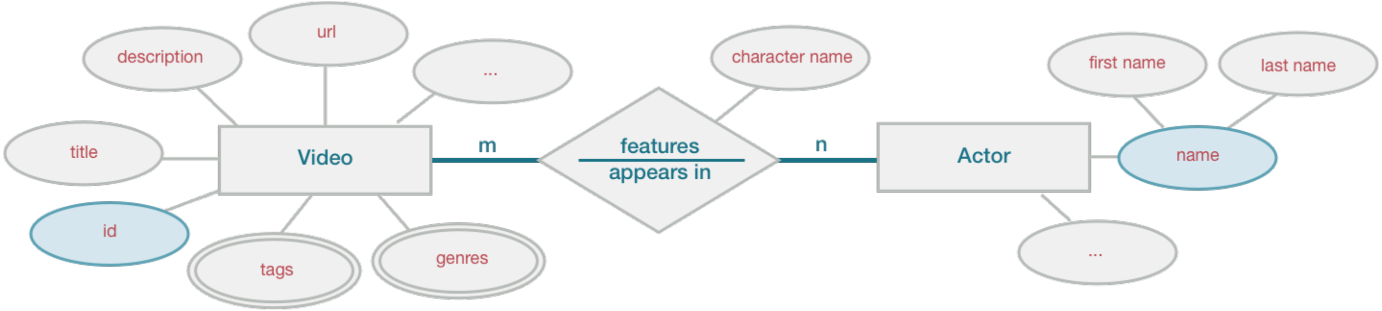
Attribute Types

- Fields to store data about an entity or relationship



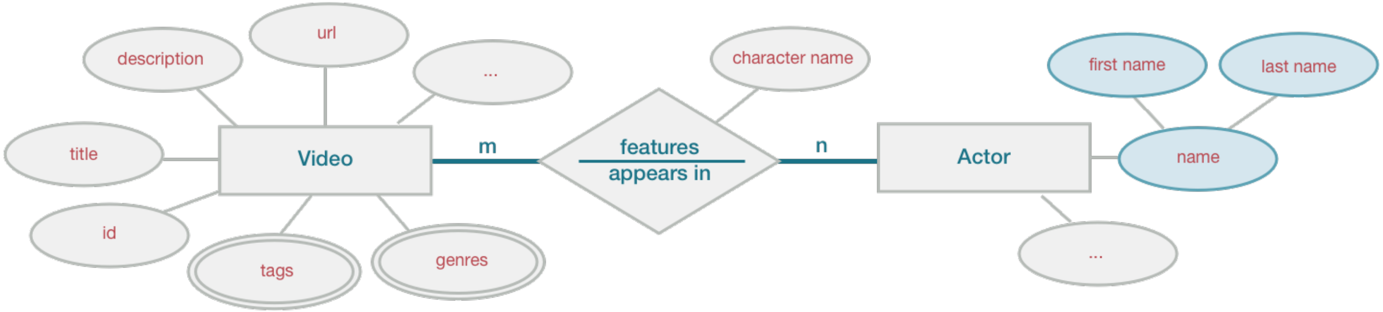
Key Attributes

- Identifies an Object



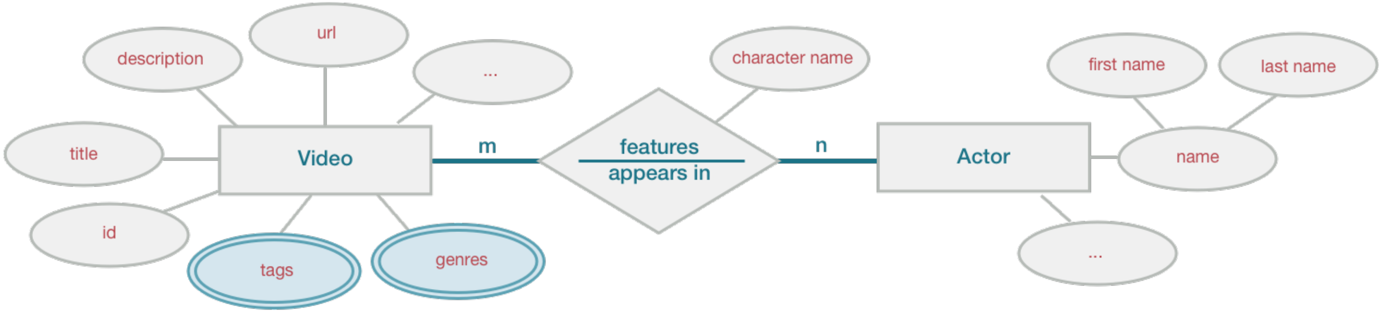
Composite Attributes

- Groups related attributes together

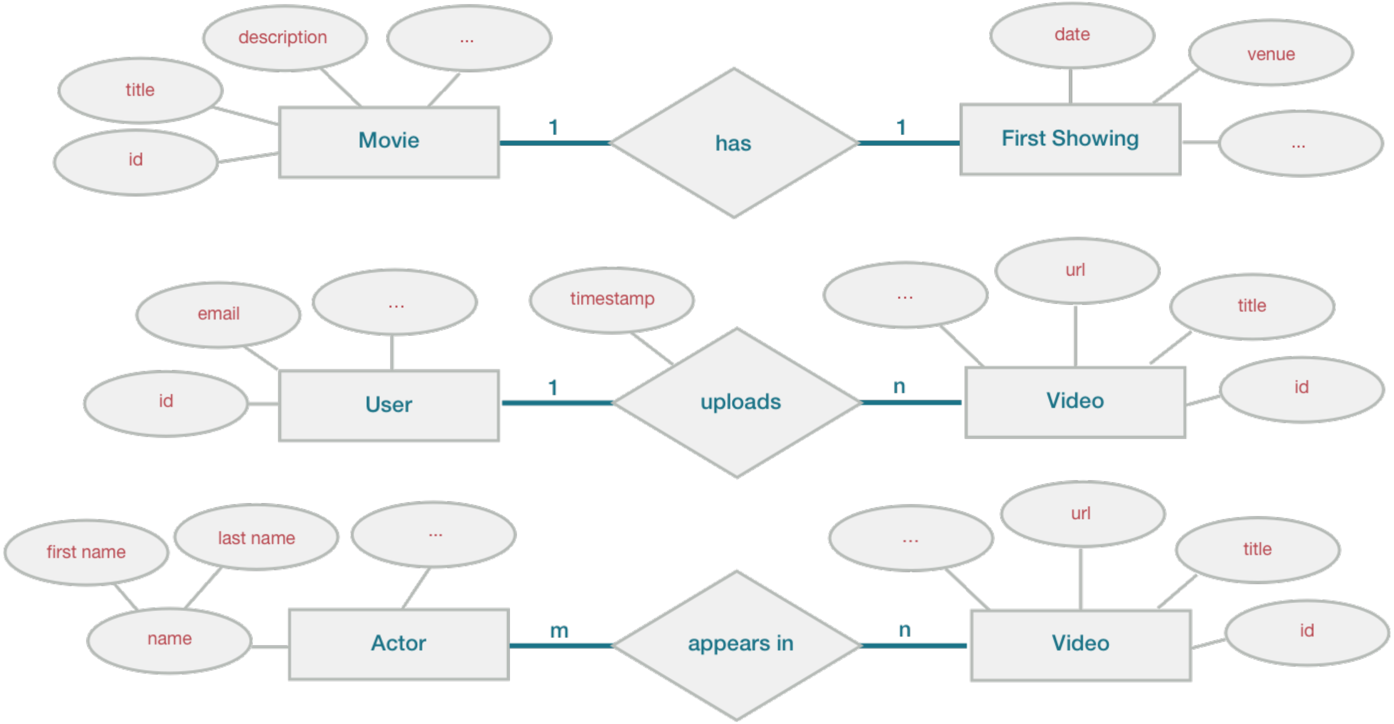


Multi Valued Attributes

- Stored multiple values per attribute

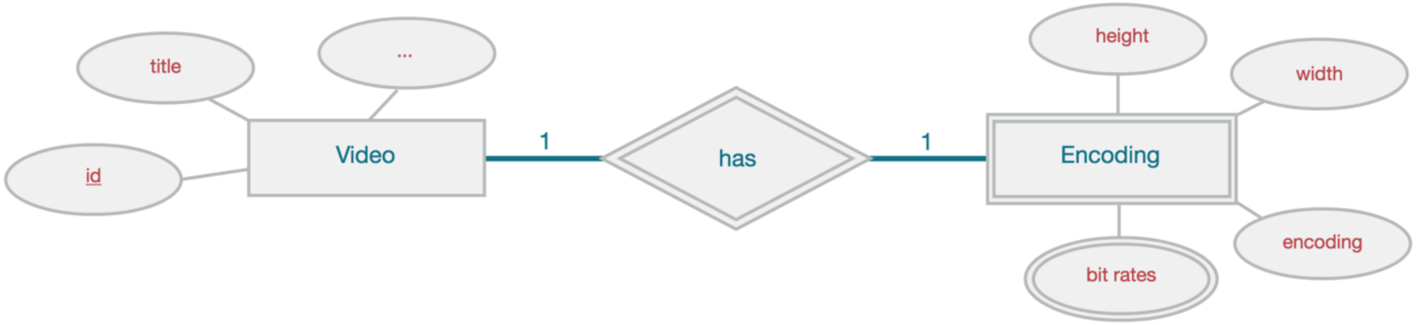


Relationship Keys



Weak Entity Types

- Entities that cannot exist without a String entity type



Exercise 03.01

DATASTAX

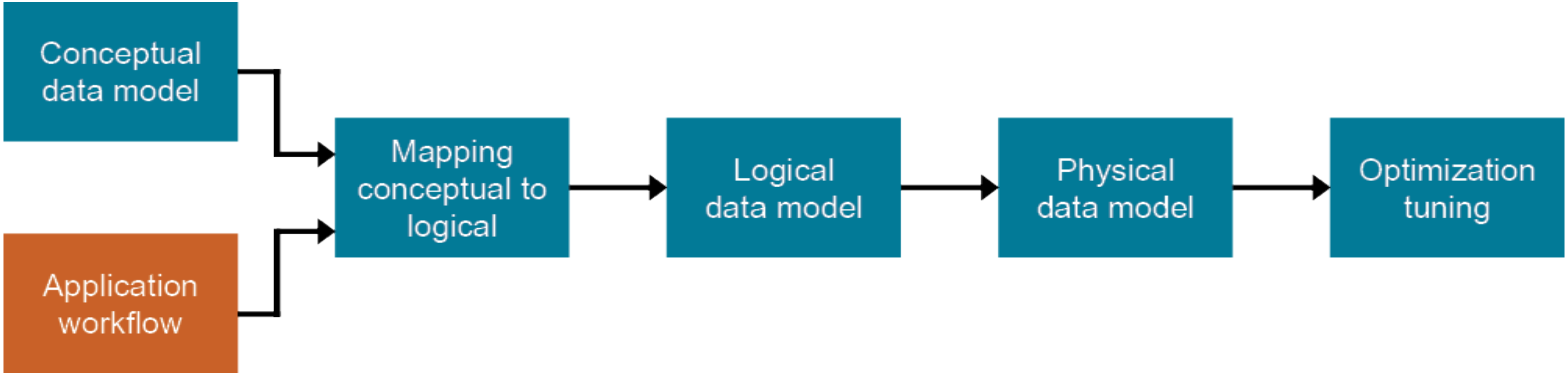
Finish the Conceptual Model

Exercise 03.01 – Finish the Conceptual Model

- Model the KillrVideo entities
- Identify the entity types, relationship types, and attribute types

Workflow and Access Patterns

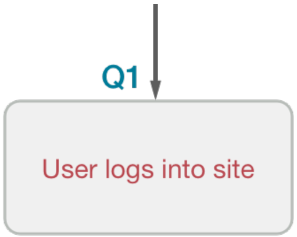
Where Are We Now?



Workflow

- Each application has a workflow—Tasks/causal dependencies form a graph
- Access patterns help determine how data is accessed—Know what queries you will run first
- Example Task:
  - Have a user login to a site

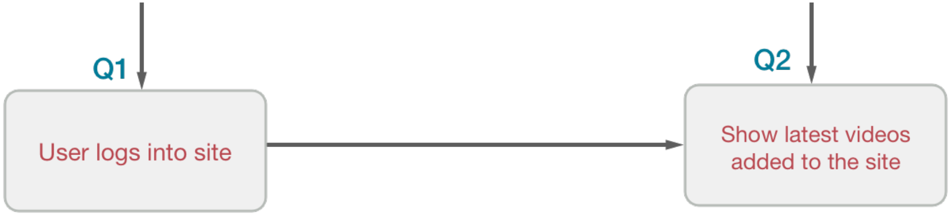
Workflow & Access Patterns



ACCESS PATTERNS

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

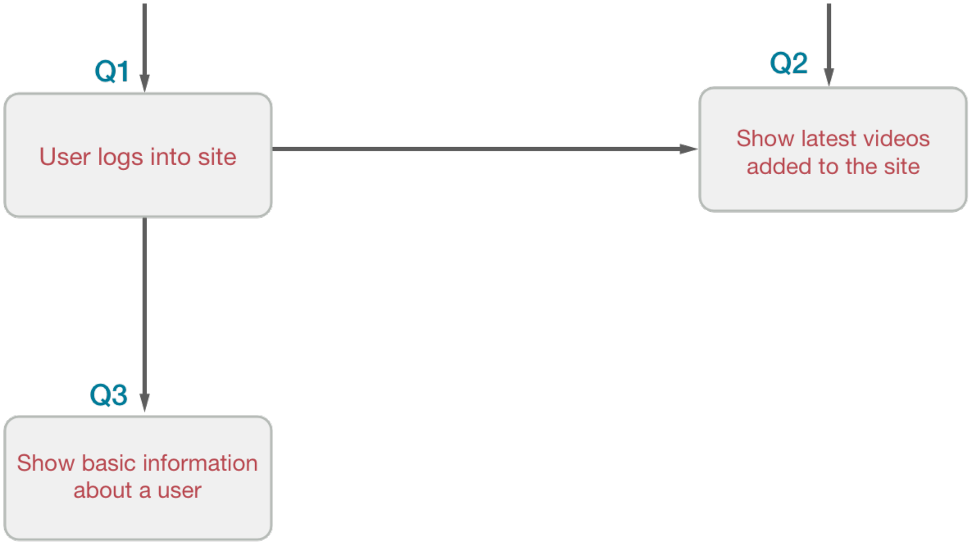
Workflow & Access Patterns



ACCESS PATTERNS

Q1: Find a user with a **specified email**
Q2: Find most recently uploaded **videos**

Workflow & Access Patterns


Q1
User logs into site
Q2
Show latest videos added to the site
Q3
Show basic information about a user

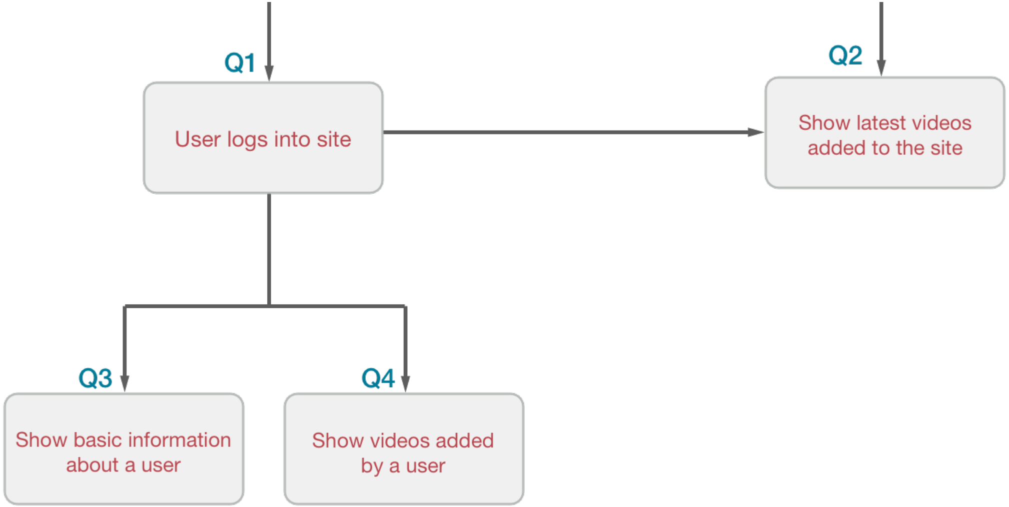
ACCESS PATTERNS

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

Q2: Find most recently uploaded **videos**

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

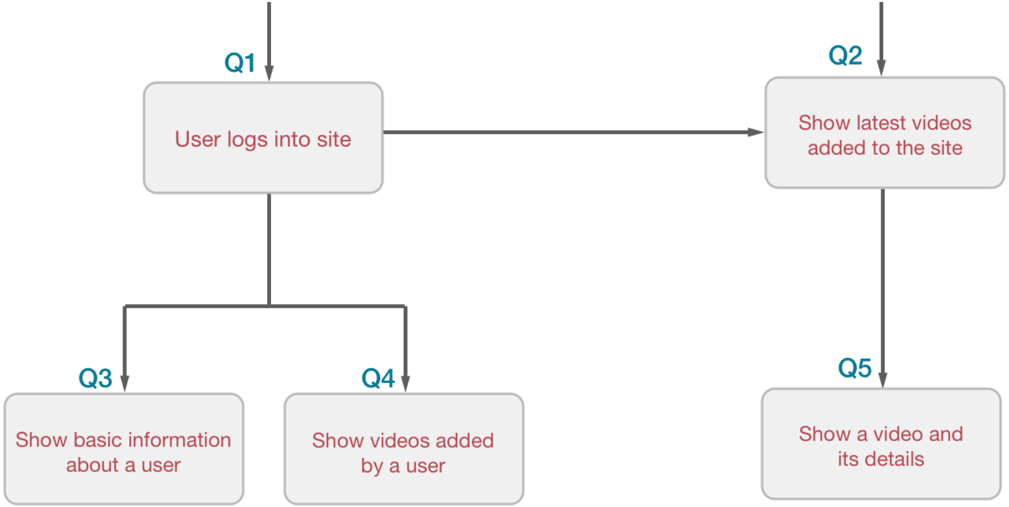
Workflow & Access Patterns


Q1
User logs into site
Q2
Show latest videos added to the site
Q3
Show basic information about a user
Q4
Show videos added by a user

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)

Workflow & Access Patterns


Q1
User logs into site
Q2
Show latest videos added to the site
Q3
Show basic information about a user
Q4
Show videos added by a user
Q5
Show a video and its details

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**

Exercise 03.02

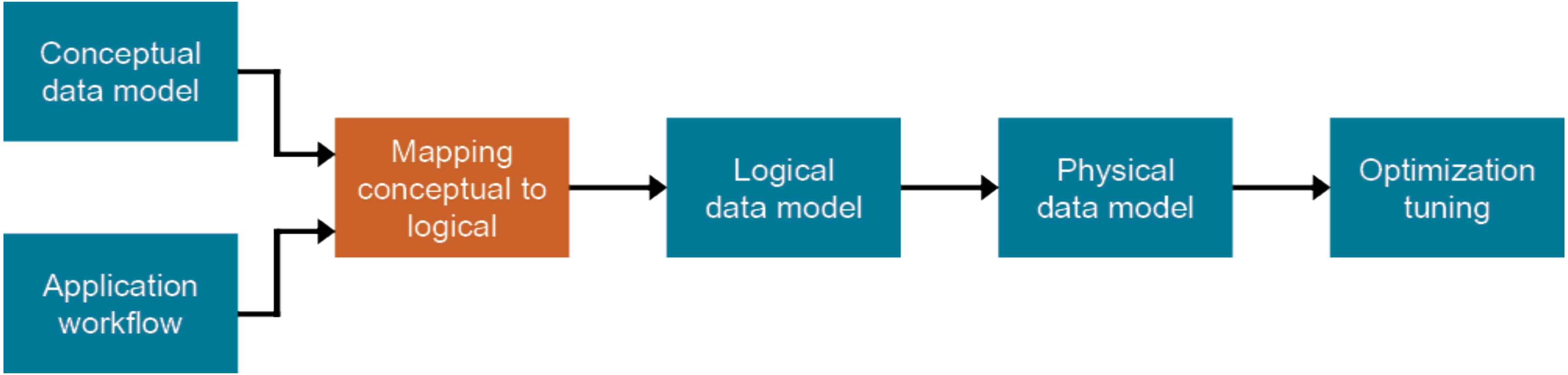
Finish the Workflow & Access Patterns

Exercise 03.02 – Finish the Workflow & Access Patterns

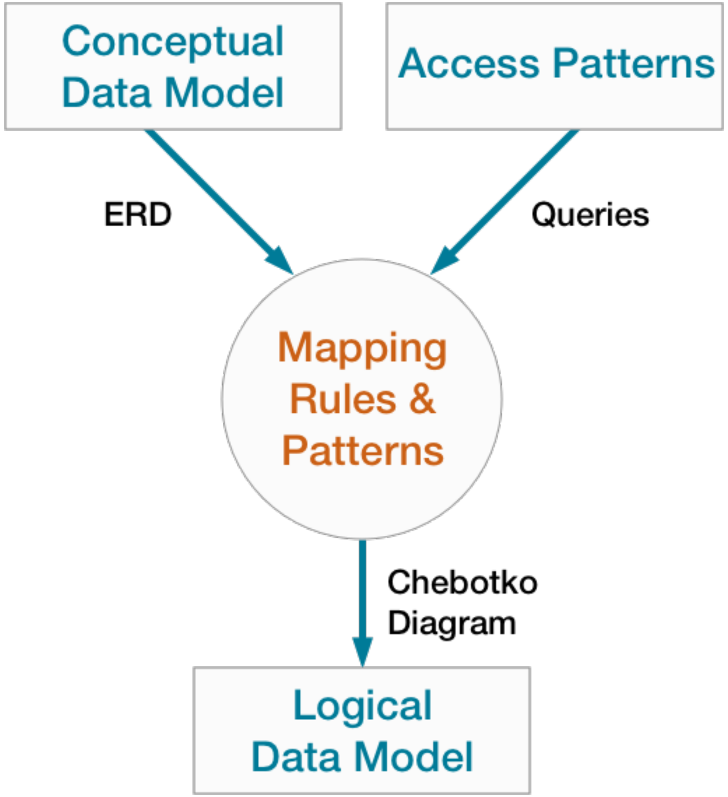
- Finish the application workflow based KillrVideo query requirements
- Add the remaining access patterns based on KillrVideo query requirements

Mapping Conceptual to Logical

Where are we now?

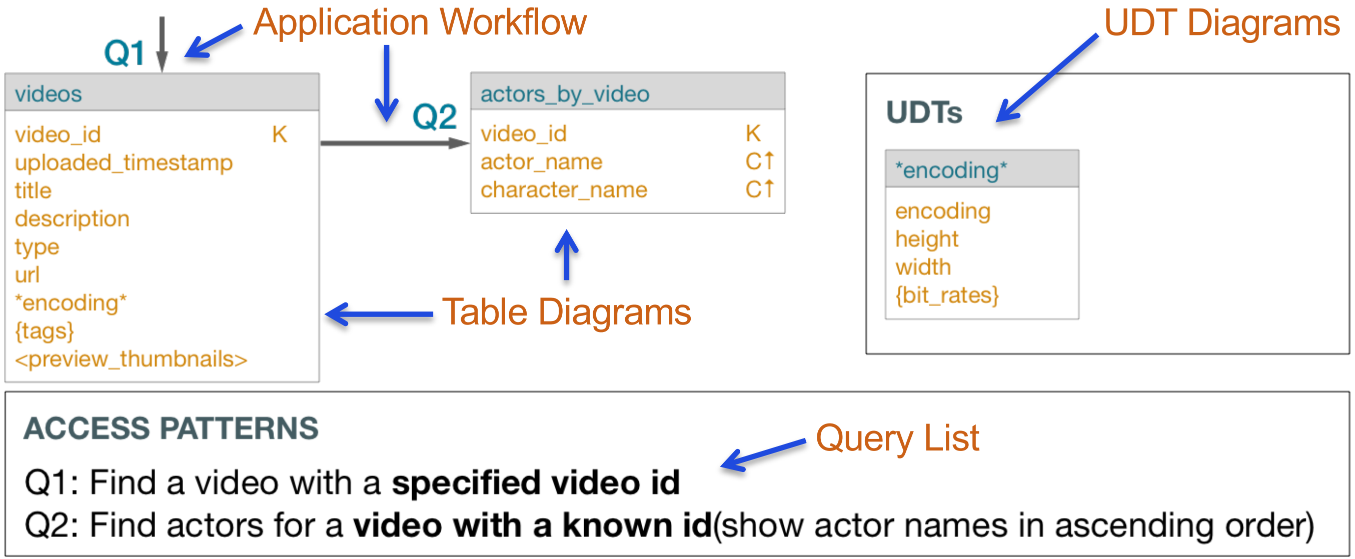


Query Driven Data Modeling



Chebotko Diagrams

- Graphical representation of Apache Cassandra^{TM} database schema design
- Documents the logical and physical data model



Application Workflow

Q1

Q2

Table Diagrams

videos	
video_id	K
uploaded_timestamp	
title	
description	
type	
url	
encoding	
{tags}	
<preview_thumbnails>	

actors_by_video	
video_id	K
actor_name	C↑
character_name	C↑

UDT Diagrams

UDTs

encoding
encoding
height
width
{bit_rates}

Query List

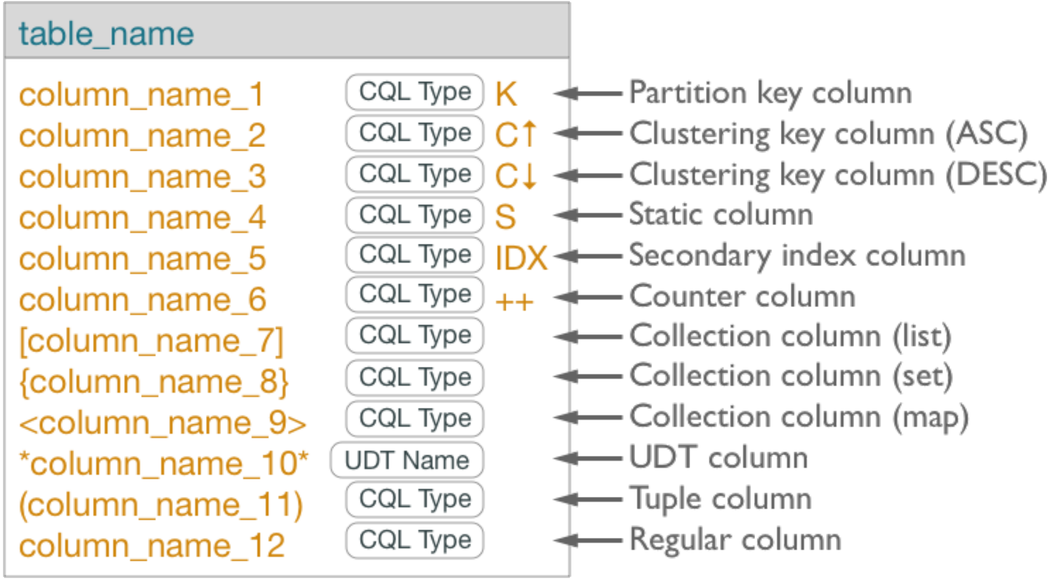
ACCESS PATTERNS

Q1: Find a video with a **specified video id**

Q2: Find actors for a **video with a known id**(show actor names in ascending order)

Chebotko Diagram Notations

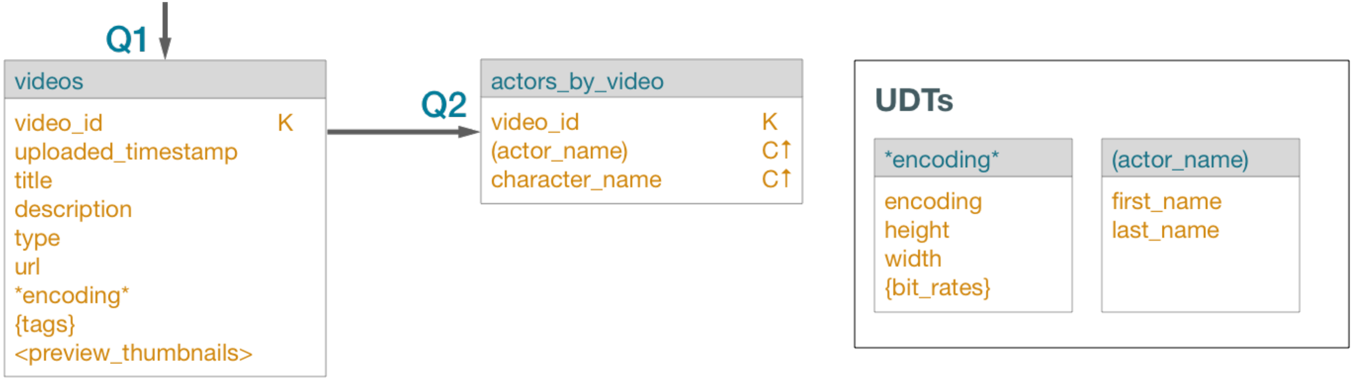
- Logical-level shows column names and properties
- Physical-level also shows the column data type



table_name			
column_name_1	CQL Type	K	Partition key column
column_name_2	CQL Type	C↑	Clustering key column (ASC)
column_name_3	CQL Type	C↓	Clustering key column (DESC)
column_name_4	CQL Type	S	Static column
column_name_5	CQL Type	IDX	Secondary index column
column_name_6	CQL Type	++	Counter column
[column_name_7]	CQL Type		Collection column (list)
{column_name_8}	CQL Type		Collection column (set)
<column_name_9>	CQL Type		Collection column (map)
column_name_10	UDT Name		UDT column
(column_name_11)	CQL Type		Tuple column
column_name_12	CQL Type		Regular column

Logical UDT Diagram

- Represents user defined types and tuples



Q1

videos

video_id K
uploaded_timestamp
title
description
type
url
encoding
{tags}
<preview_thumbnails>

Q2

actors_by_video

video_id K
(actor_name) C↑
character_name C↑

UDTs

encoding

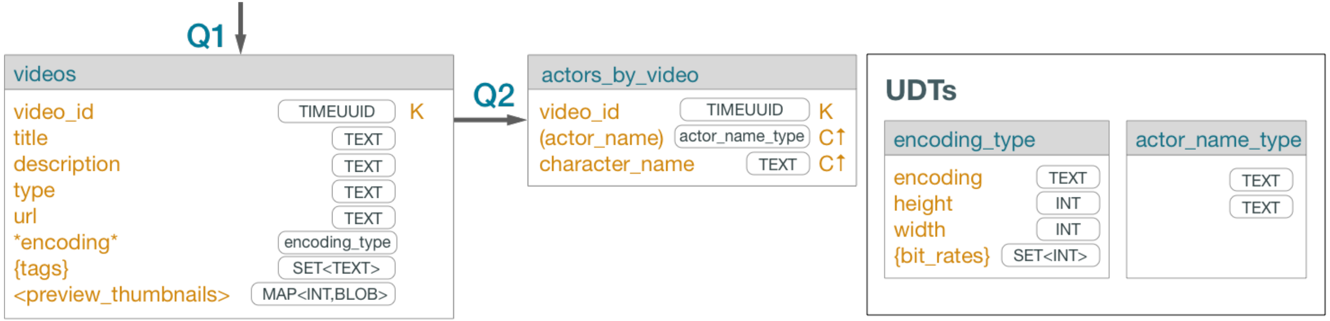
encoding
height
width
{bit_rates}

(actor_name)

first_name
last_name

Physical UDT Diagram

- Represents user defined types and tuples



videos		
video_id	TIMEUUID	K
title	TEXT	
description	TEXT	
type	TEXT	
url	TEXT	
encoding	encoding_type	
{tags}	SET<TEXT>	
<preview_thumbnails>	MAP<INT,BLOB>	

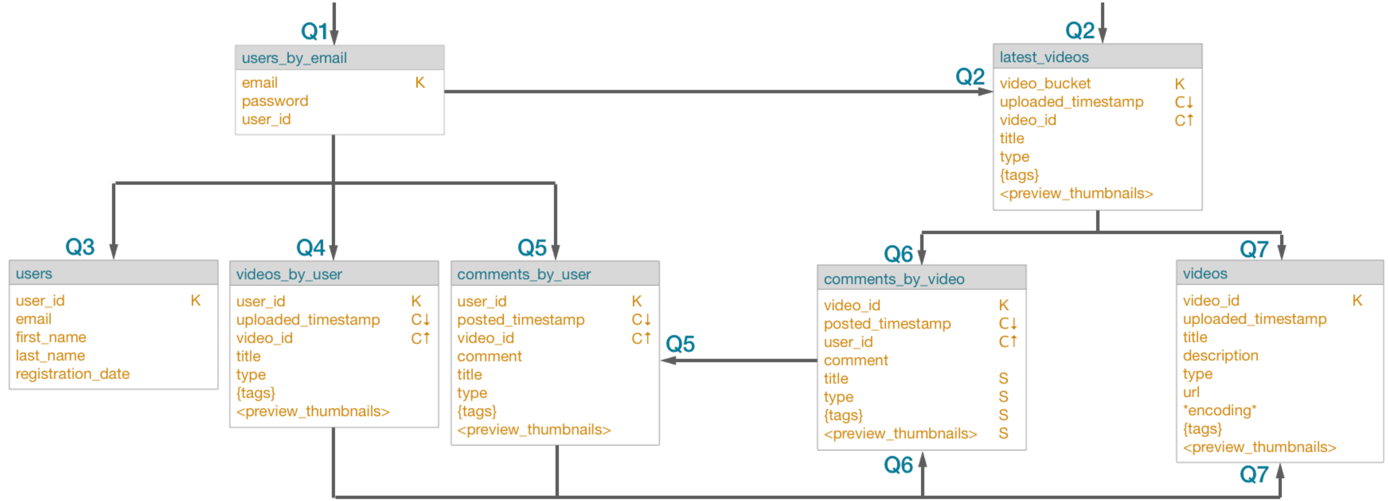
actors_by_video		
video_id	TIMEUUID	K
(actor_name)	actor_name_type	C↑
character_name	TEXT	C↑

UDTs

encoding_type	
encoding	TEXT
height	INT
width	INT
{bit_rates}	SET<INT>

actor_name_type	
	TEXT
	TEXT

Example Chebotko Diagram



users_by_email	
email	K
password	
user_id	

latest_videos	
video_bucket	K
uploaded_timestamp	C↓
video_id	C↑
title	
type	
{tags}	
<preview_thumbnails>	

users	
user_id	K
email	
first_name	
last_name	
registration_date	

videos_by_user	
user_id	K
uploaded_timestamp	C↓
video_id	C↑
title	
type	
{tags}	
<preview_thumbnails>	

comments_by_user	
user_id	K
posted_timestamp	C↓
video_id	C↑
comment	
title	
type	
{tags}	
<preview_thumbnails>	

comments_by_video	
video_id	K
posted_timestamp	C↓
user_id	C↑
comment	
title	S
type	S
{tags}	S
<preview_thumbnails>	S

videos	
video_id	K
uploaded_timestamp	
title	
description	
type	
url	
encoding	
{tags}	
<preview_thumbnails>	

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 comments posted by a **user with a known id**(show most recently posted comments first)
Q6: Find comments posted for a **video with a known id**(show most recent comments first)
Q7: Find a video with a **specified video id**

UDTs

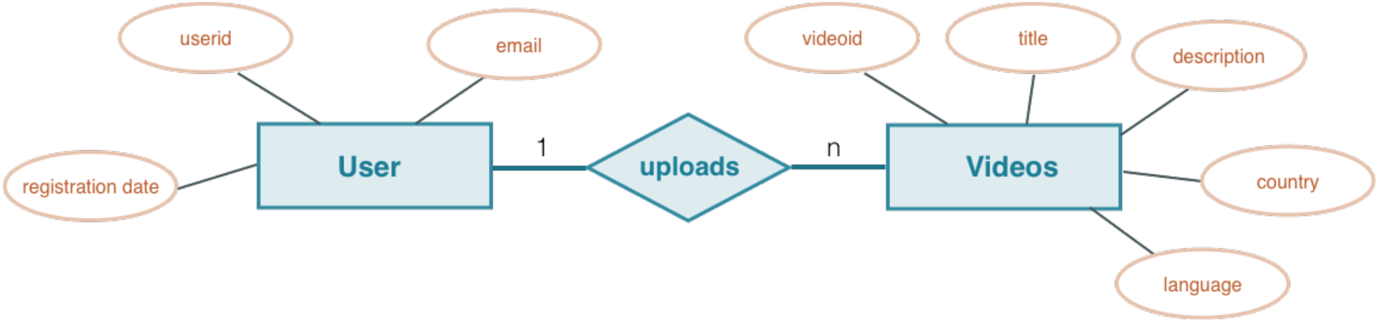
encoding
encoding
height
width
{bit_rates}

Data Modeling Principles

- Know your data
- Know your queries
- Nest data
- Duplicate data

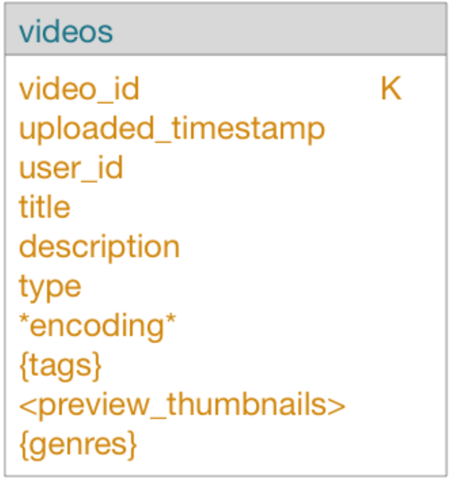
Know Your Data

- Data captured by conceptual data model
- Define what is stored in database
- Preserve properties so that data is organized correctly

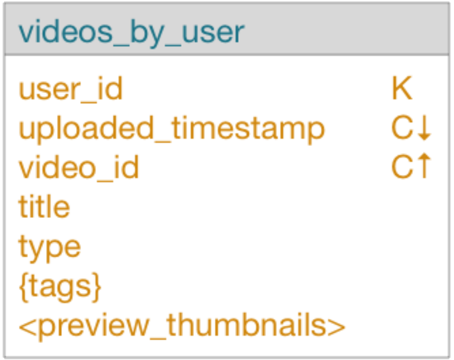


Know your Data

- Entity and relationship keys affect the table primary keys
- Primary key uniquely identifies a row / entity / relationship
- Composed of a key and possibly additional columns

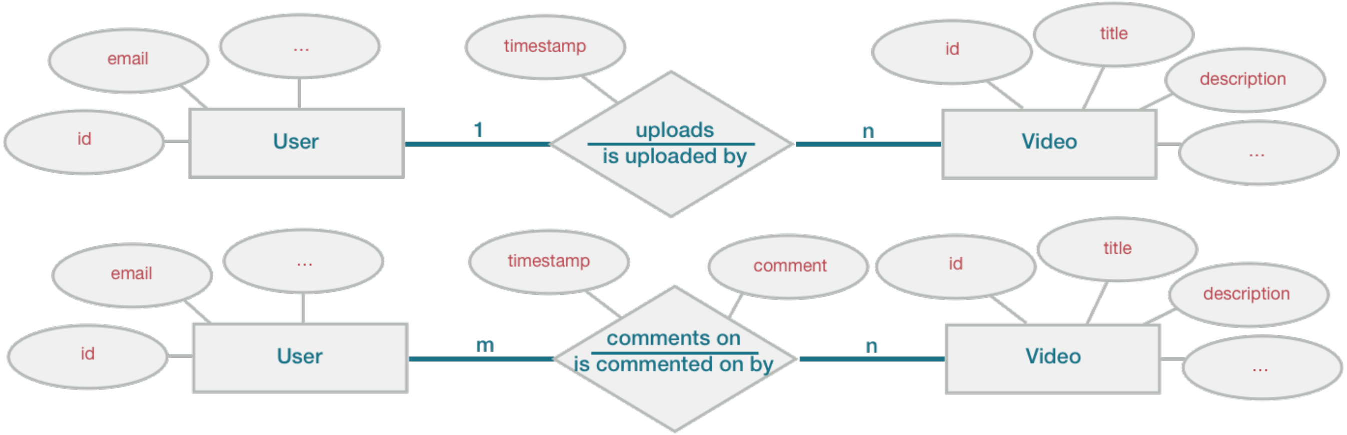


videos	
video_id	K
uploaded_timestamp	
user_id	
title	
description	
type	
encoding	
{tags}	
<preview_thumbnails>	
{genres}	



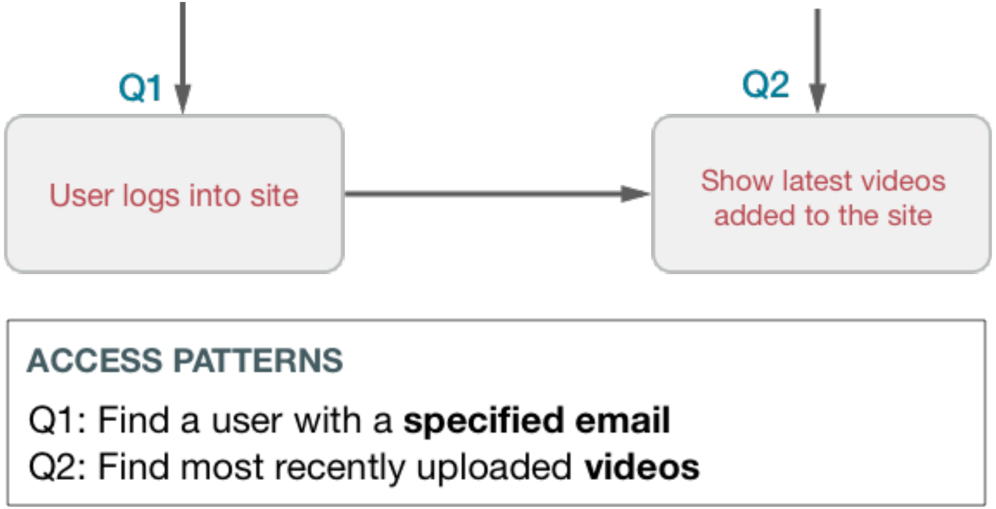
videos_by_user	
user_id	K
uploaded_timestamp	C↓
video_id	C↑
title	
type	
{tags}	
<preview_thumbnails>	

Cardinality Constraints Affect the Key for Relationships



Know your Queries

- Queries captured by application workflow model
- Table schema design changes if queries change

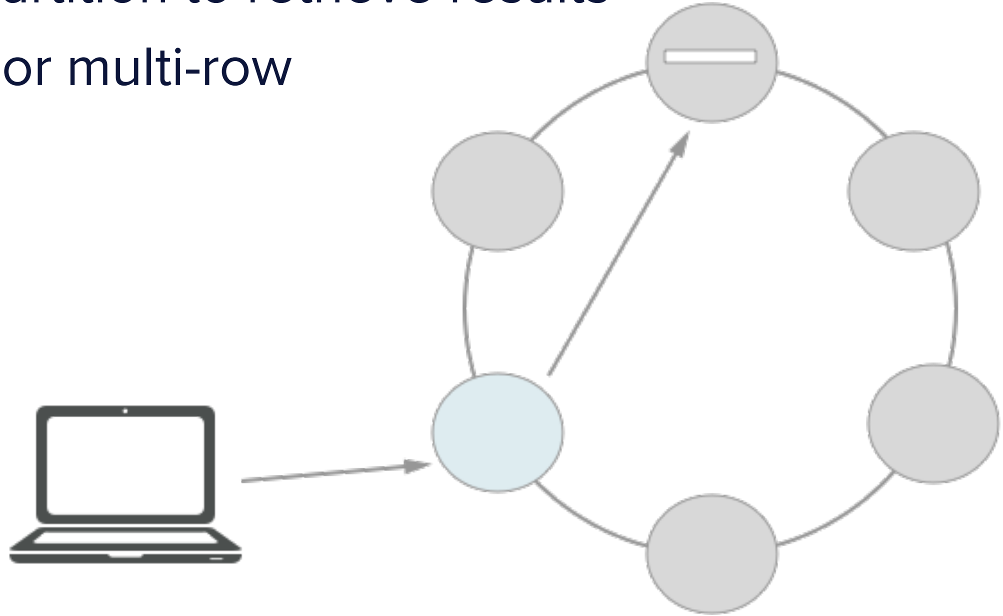


ACCESS PATTERNS

Q1: Find a user with a **specified email**
Q2: Find most recently uploaded **videos**

Single Partition per Query = Ideal

- Most efficient access pattern
- Query accesses only one partition to retrieve results
- Partition can be single-row or multi-row



Partition+ Per Query = Acceptable

- Less efficient access pattern but not necessarily bad
- Query needs to access multiple partitions to retrieve results

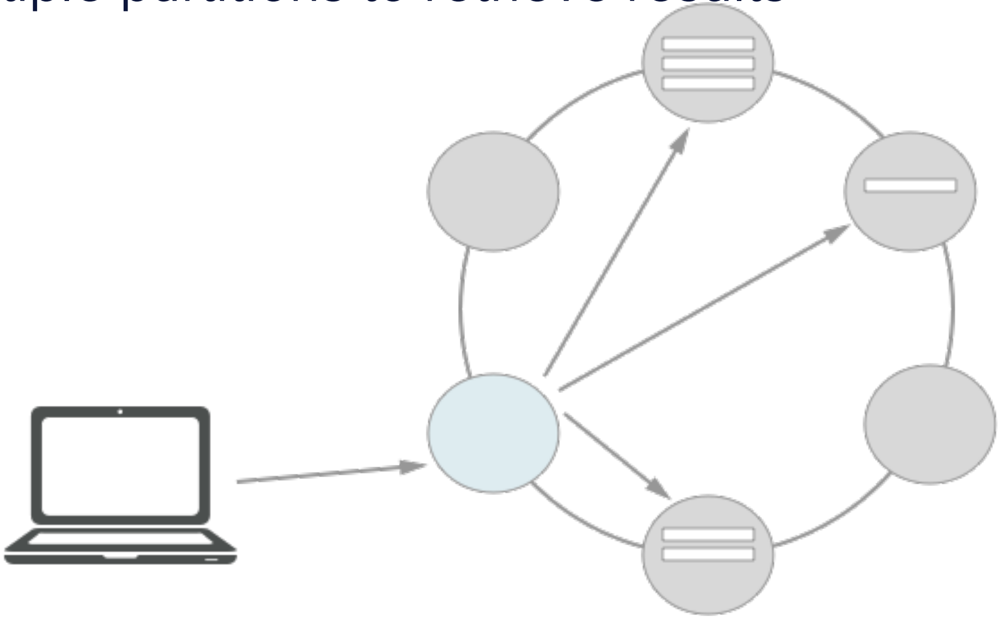
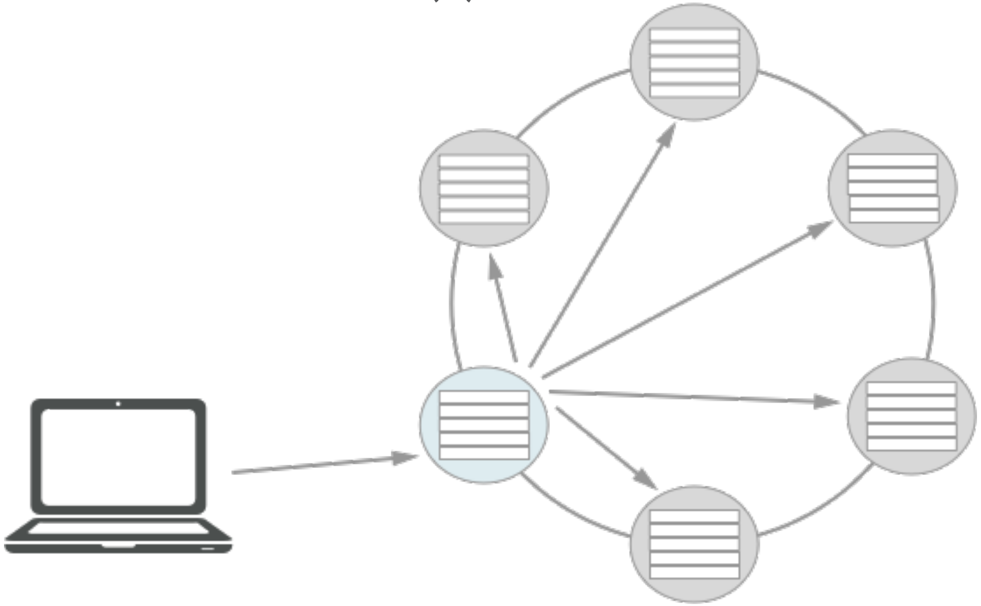


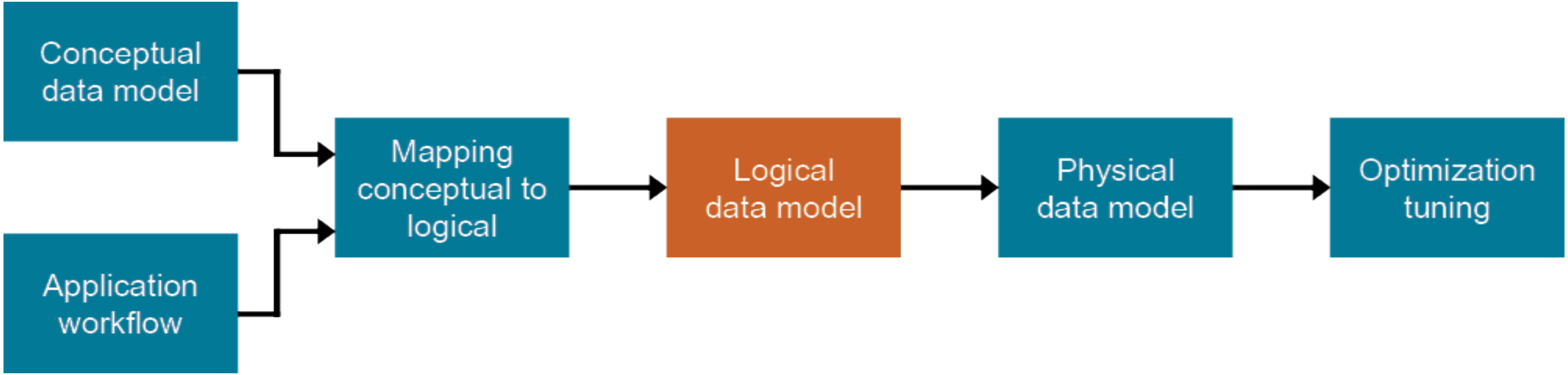
Table Scan/Multi-Table Scan = Anti-Pattern

- Least efficient type of query but may be needed in some cases
- Query needs to access all partitions in a table(s) to retrieve results



Logical Model

Where Are We Now?

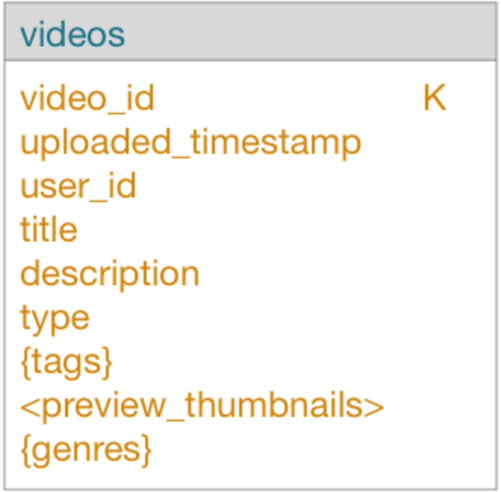


Nest Data

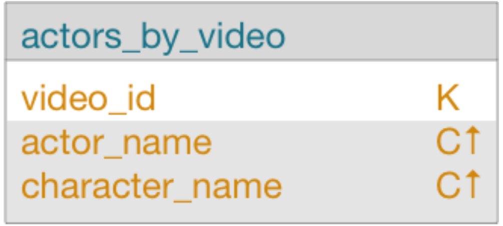
- Data nesting is the main data modeling technique
- Nesting organizes multiple entities into a single partition
- Supports partition per query data access
- Three data nesting mechanisms:
  - Clustering columns – multi-row partitions
  - Collection columns
  - User-defined type columns

Nest Data - Clustering Columns

- Clustering column—primary data nesting mechanism
- Partition key identifies an entity that other entities will nest into
- Values in a clustering column identify the nested entities
- Multiple clustering columns implement multi-level nesting



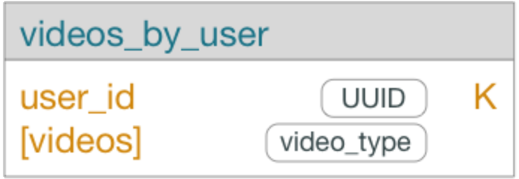
videos	
video_id	K
uploaded_timestamp	
user_id	
title	
description	
type	
{tags}	
<preview_thumbnails>	
{genres}	



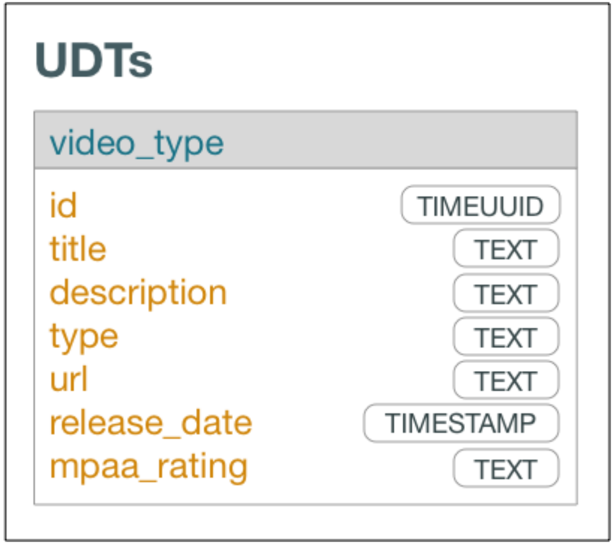
actors_by_video	
video_id	K
actor_name	C↑
character_name	C↑

Nest Data – UDT

- User-defined type—secondary data nesting mechanism
- Represents 1-1 relationship, but can use in conjunction with collections
- Easier than working with multiple collection columns



videos_by_user		
user_id	UUID	K
[videos]	video_type	

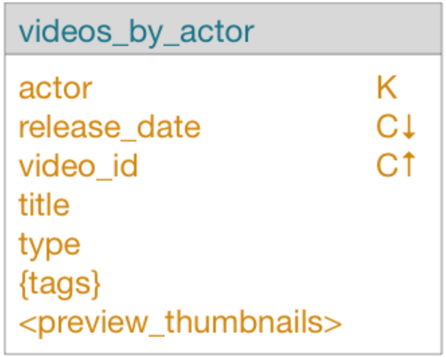


UDTs

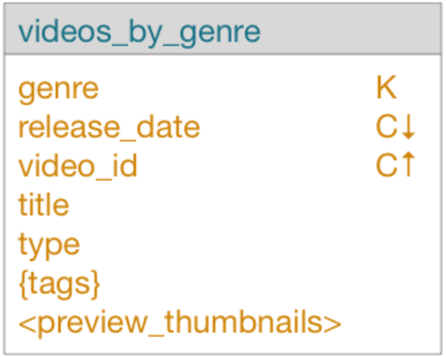
video_type	
id	TIMEUUID
title	TEXT
description	TEXT
type	TEXT
url	TEXT
release_date	TIMESTAMP
mpaa_rating	TEXT

Duplicate Data

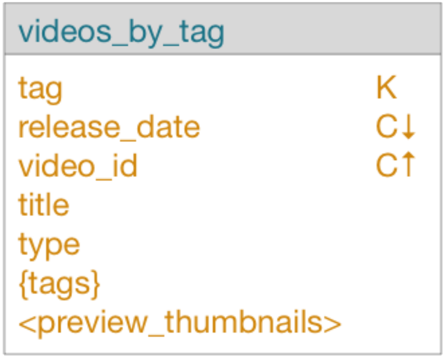
- Partition per query and data nesting may result in data duplication
- Query results are pre-computed and materialized
- Data can be duplicated across tables, partitions, and / or rows



videos_by_actor	
actor	K
release_date	C↓
video_id	C↑
title	
type	
{tags}	
<preview_thumbnails>	



videos_by_genre	
genre	K
release_date	C↓
video_id	C↑
title	
type	
{tags}	
<preview_thumbnails>	



videos_by_tag	
tag	K
release_date	C↓
video_id	C↑
title	
type	
{tags}	
<preview_thumbnails>	

Mapping Rules

Mapping Rules for Query Driven Methodology

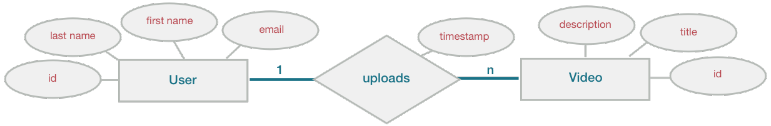
- Mapping rules ensure that a logical data model is correct
- Each query has a corresponding table
- Tables are designed to allow queries to execute properly
- Tables return data in the correct order

What are the Rules?

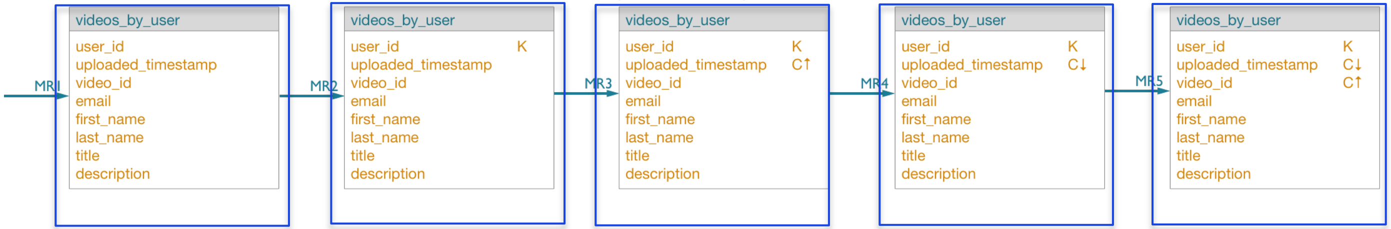
- Mapping Rule 1: Entities and relationships
- Mapping Rule 2: Equality search attributes
- Mapping Rule 3: Inequality search attributes
- Mapping Rule 4: Ordering attributes
- Mapping Rule 5: Key attributes

Mapping Rules in Action

- Create a table schema from the conceptual data model and for each query
- Apply the mapping rules in order



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



videos_by_user	
user_id	
uploaded_timestamp	
video_id	
email	
first_name	
last_name	
title	
description	

Model entities and relationships

videos_by_user	
user_id	K
uploaded_timestamp	
video_id	
email	
first_name	
last_name	
title	
description	

Equality search attributes

videos_by_user	
user_id	K
uploaded_timestamp	C↑
video_id	
email	
first_name	
last_name	
title	
description	

Inequality search attributes

videos_by_user	
user_id	K
uploaded_timestamp	C↓
video_id	
email	
first_name	
last_name	
title	
description	

Ordering attributes

videos_by_user	
user_id	K
uploaded_timestamp	C↓
video_id	C↑
email	
first_name	
last_name	
title	
description	

Key attributes

Exercise 03.03

DATASTAX®

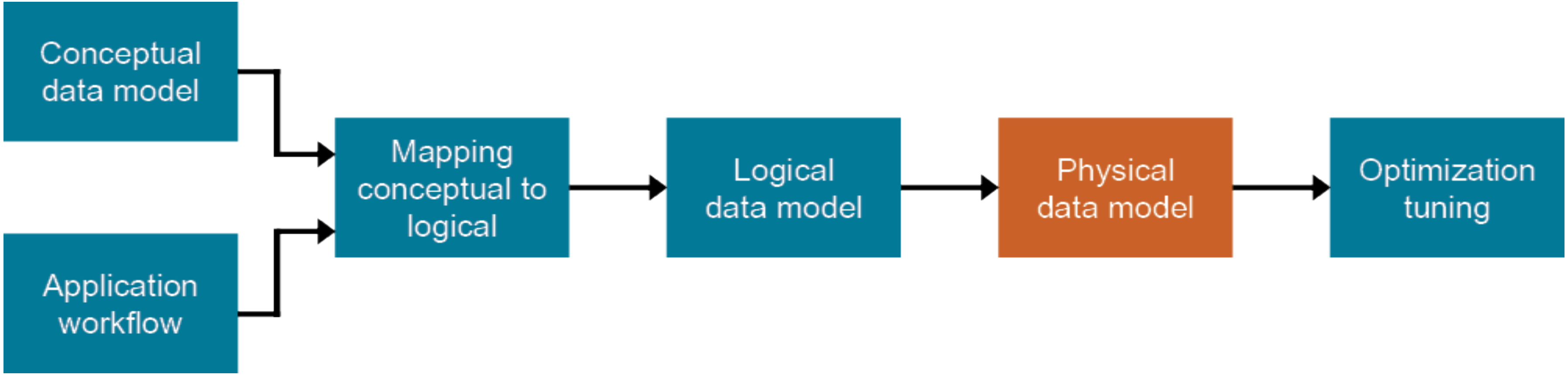
Finalize your Logical Model

Exercise 03.03 – Finalize your Logical Model

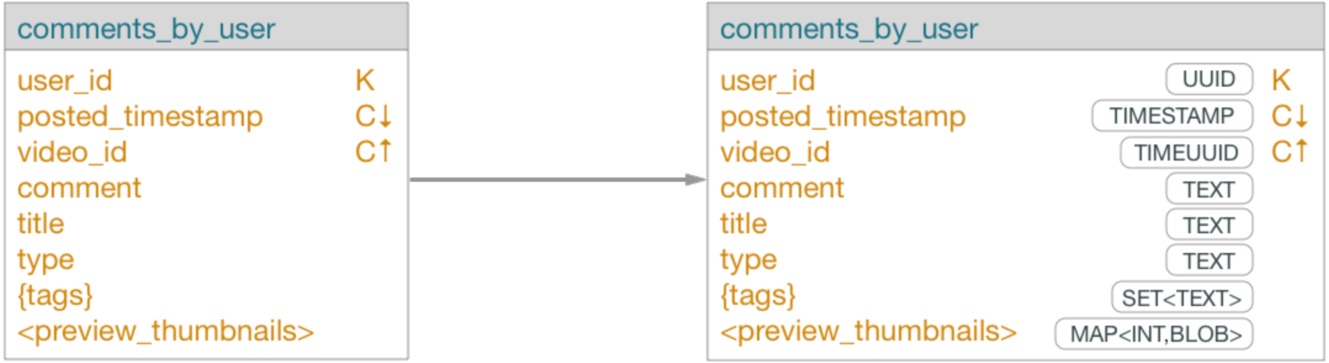
- Add tables to the logical model to support additional queries
- Ensure that the mapping rules are applied appropriately

Physical Model

Where Are We Now?



Adding Data Types



comments_by_user	
user_id	K
posted_timestamp	C↓
video_id	C↑
comment	
title	
type	
{tags}	
<preview_thumbnails>	

comments_by_user		
user_id	UUID	K
posted_timestamp	TIMESTAMP	C↓
video_id	TIMEUUID	C↑
comment	TEXT	
title	TEXT	
type	TEXT	
{tags}	SET<TEXT>	
<preview_thumbnails>	MAP<INT,BLOB>	

Creating Tables

```
CREATE TABLE comments_by_user (
    user_id UUID,
    posted_timestamp TIMESTAMP,
    video_id TIMEUUID,
    comment TEXT,
    title TEXT,
    type TEXT,
    tags SET<TEXT>,
    preview_thumbnails MAP<INT, BLOB>,
    PRIMARY KEY ((user_id), posted_timestamp, video_id)
) WITH CLUSTERING ORDER BY (posted_timestamp DESC, video_id ASC);
```

Data Loading Methods

- COPY Command
- SSTable Loader
- Spark for Data Loading
- More about loading in Module 5!

Basic Data Loading with CQLCOPY

- COPY TO exports data from a table to a CSV file
- COPY FROM imports data to a table from a CSV file
- The process verifies the PRIMARY KEY and updates existing records
- If HEADER = false is specified the fields are imported in deterministic order
- When column names are specified, fields are imported in that order—missing and empty fields set to null
- Source cannot have more fields than the target table--can have fewer fields

```
COPY table1 (column1, column2, column3) FROM 'table1data.csv'
WITH HEADER=true;
```

Exercise 03.04

DATASTAX

Finalize your Physical Model

Exercise 03.04 – Finalize your Physical Model

- Add data types to the physical data model
- Run the `CQL CREATE TABLE` statements for each table in physical model
- Load data and run some queries to test the physical data model