cassandra

Lab2 - Cassandra

Data Structuring and NoSQL Databases

Larbi Boubchir

CREATE THE KEYSPACE

```
CREATE KEYSPACE demoVideo
WITH REPLICATION = {
        'class': 'SimpleStrategy',
        'replication_factor' : 1
};

USE demoVideo;
```

CREATION OF THE TABLE

```
CREATE TABLE videos (
    id int,
    name text,
    runtime int,
    year int,
    PRIMARY KEY (id)
);
```

INSERTION

- Insert this data into a video table
- Either directly with the INSERT clause or by using a CSV file and the COPY clause

id	name	runtime	year
1	Insurgent	119	2015
2	Interstellar	98	2014
3	Mockingjay	122	2014

QUERYING

- How many rows have been inserted ?
- View All Records
- Show information about the video «insurgent»
- Show videos with year greater than 2014. What do you get? Why?

```
 Number of Rows
----------------
              3

(1 rows)
```

```
 id | name         | runtime | year
----+--------------+---------+------
  1 |    Insurgent |     119 | 2015
  2 | Interstellar |      98 | 2014
  3 |   Mockingjay |     122 | 2014

(3 rows)
```

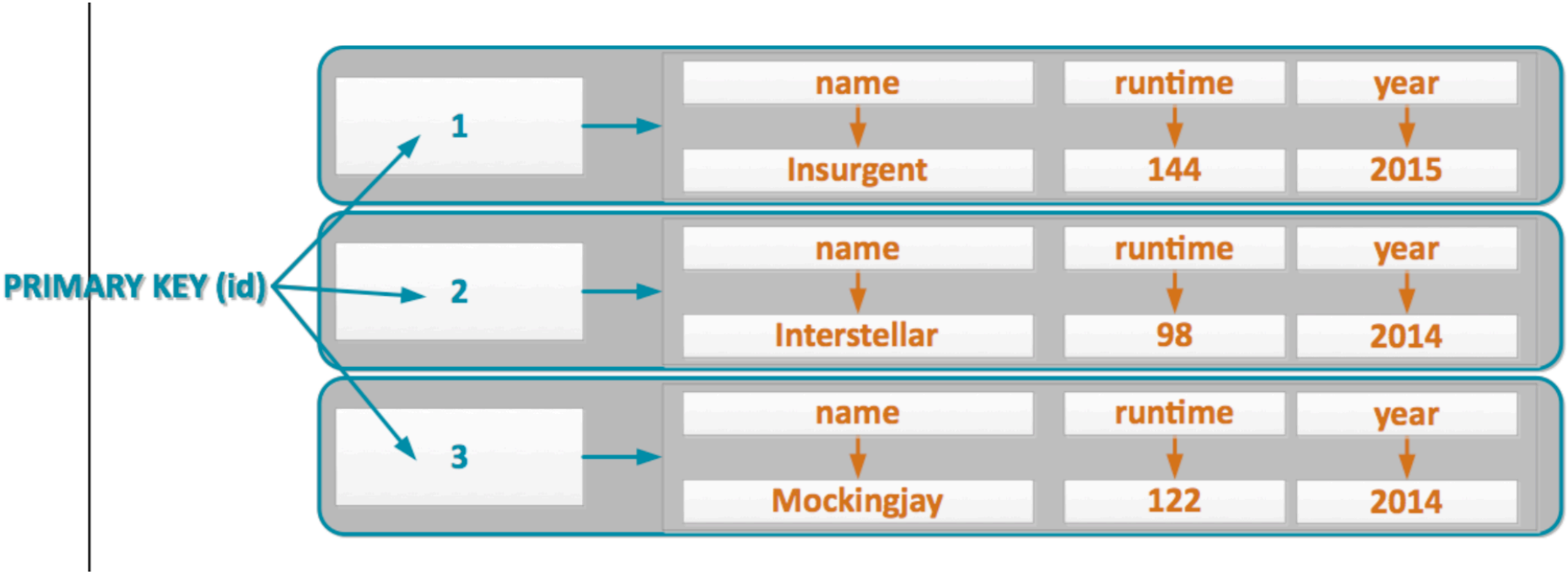
```
 id | name      | runtime | year
----+-----------+---------+------
  1 | Insurgent |     119 | 2015

(1 rows)
```

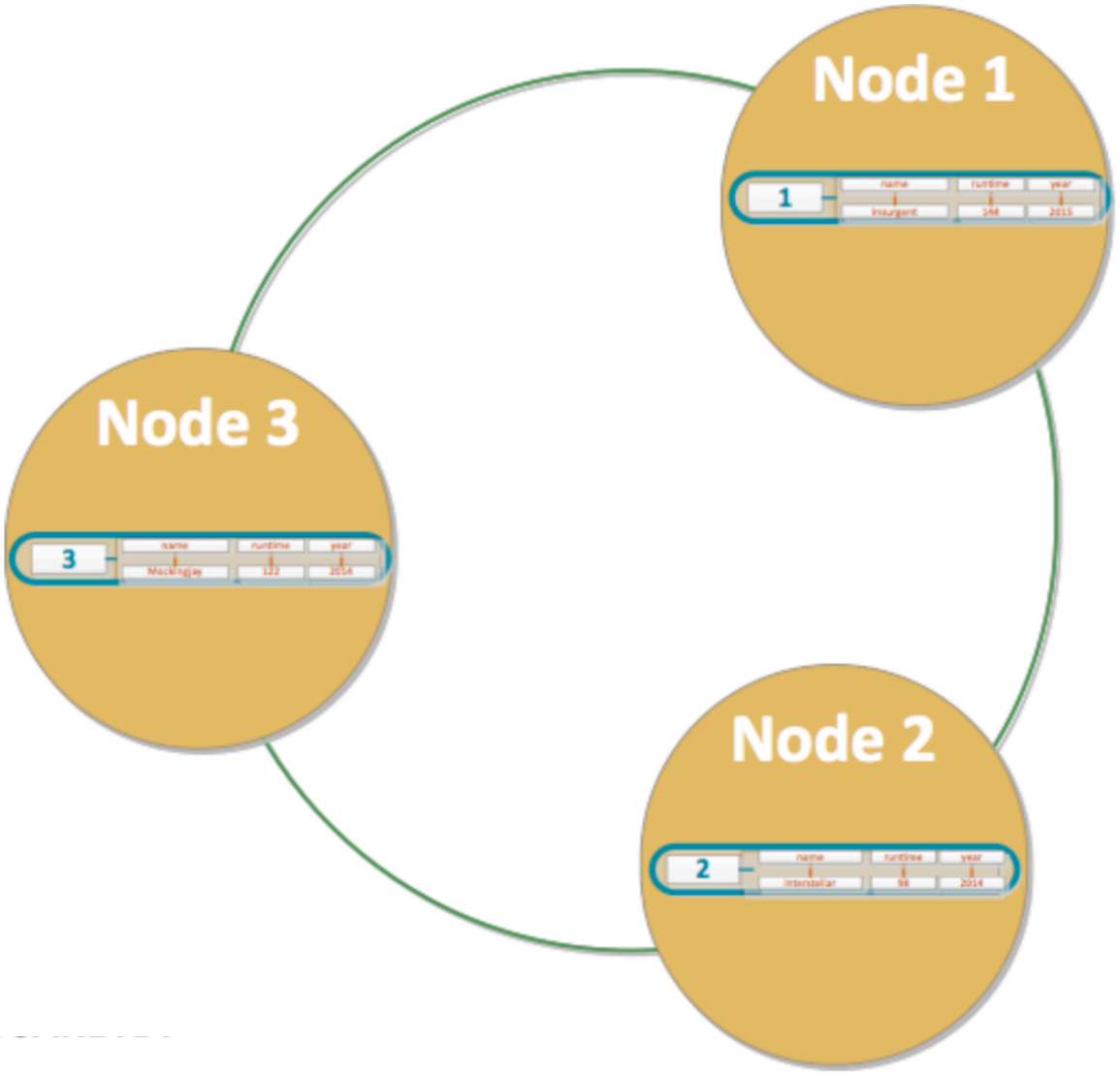
For filtering columns in Cassandra, indexe needs to be created.

Syntax: Create index IndexName on KeyspaceName.TableName(ColumnName);

PHYSICAL STORAGE

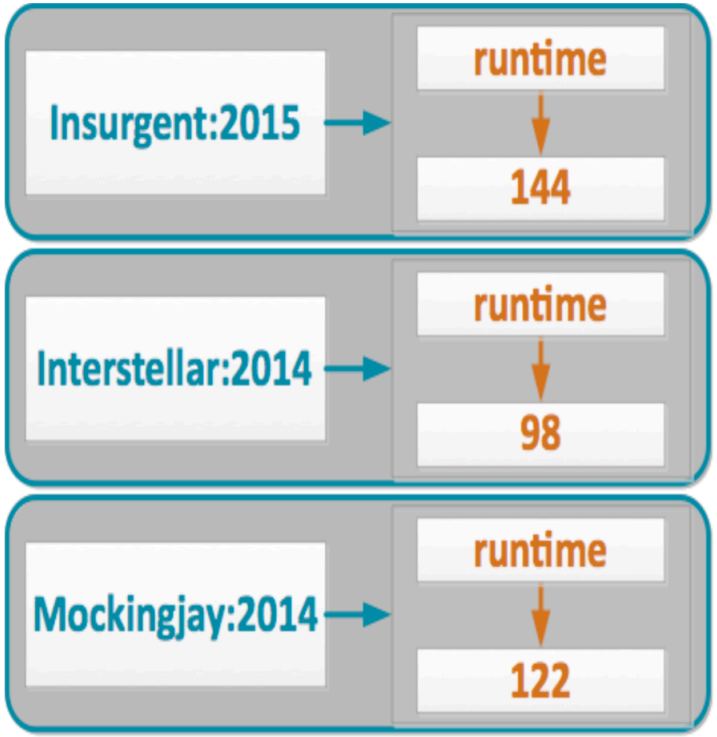


PARTITIONED STORAGE



IS THIS A SOLUTION? TRY IT

```
CREATE TABLE videos_by_name_year (
name text,
runtime int,
year int,
PRIMARY KEY ((name, year))
);
```



QUERIES

- Find the movie "Insurgent" made in 2015
- Find information about the film "Interstellar"
- What are the films made in 2014?

CASSANDRA-UPSERTS

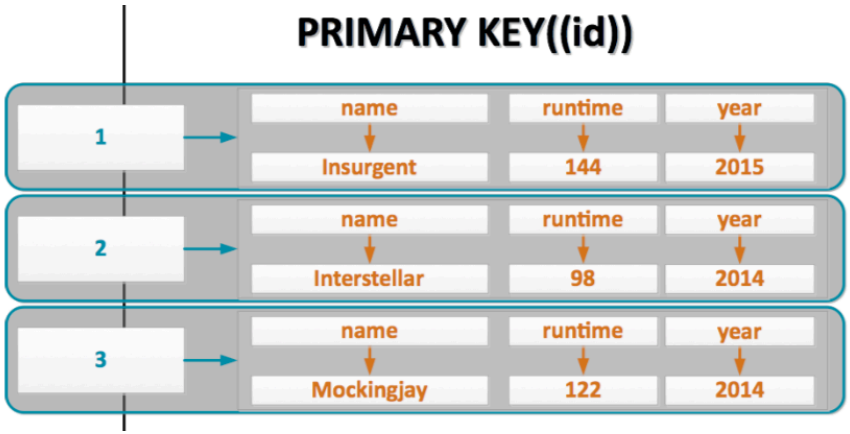
```
INSERT INTO videos_by_name_year (name , year , runtime)
VALUES ('Insurgent',2015, 127) ;
```

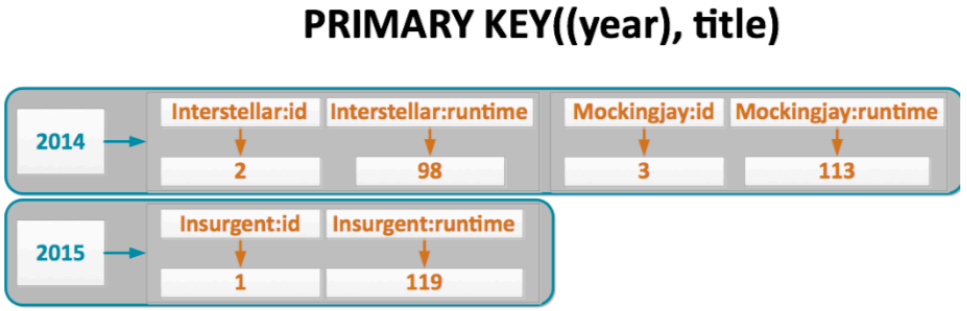
```
SELECT count(*) from videos_by_name_year
```

What’s happen?

CLUSTERING COLUMNS

```
CREATE TABLE videos_by_year (
id int,
name text,
runtime int,
year int,
PRIMARY KEY ((year), name )
);
```





CLUSTERING COLUMN WITH ORDER

Default ascending order

If we want to specify a descending order:

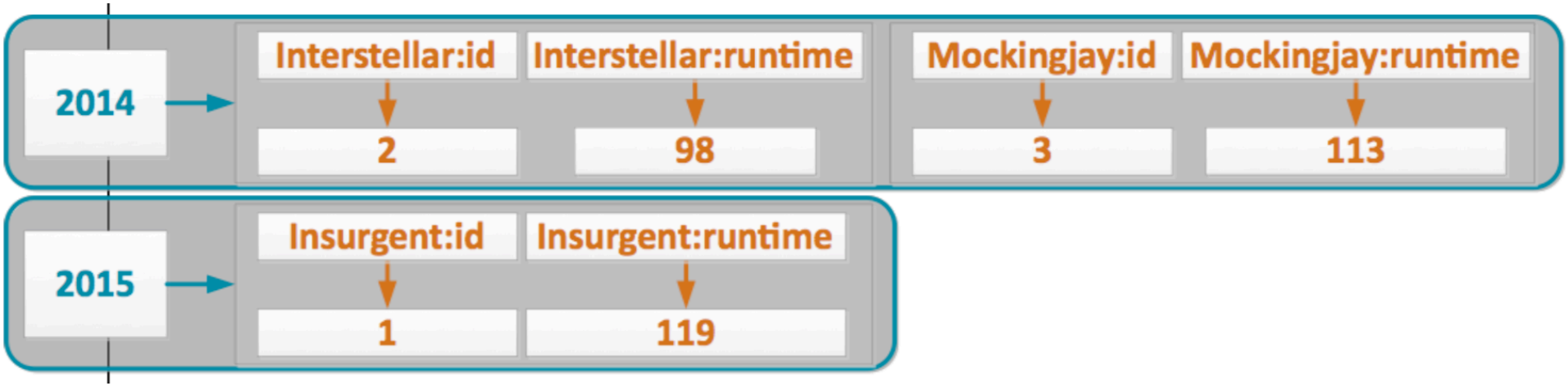
```
CREATE TABLE videos_by_year (
id int,
name text,
runtime int,
year int,
PRIMARY KEY ((year), name) )
WITH CLUSTERING ORDER BY (name DESC);
```

QUERYING CLUSTERING COLUMNS

SELECT * FROM videos_by_year WHERE year = 2014 AND name = 'Mockingjay';

Or (comparison operator)

SELECT * FROM videos_by_year WHERE year = 2014 AND name >= 'Interstellar';



ALTER TABLE

ALTER TABLE table1 ADD another_column text;

ALTER TABLE table1 DROP another_column;

- The column PRIMARY KEY can’t be modified

- Delete data

TRUNCATE table1;

MULTI-VALUED COLUMN

A column can contain several values (unlike RDBMS)

SET <TEXT> collection of typed and ordered values (depending on value)

LIST <TEXT> ordered by position

MAP <TEXT, INT> key-value collection ordered by key

UDT (USER DEFINED TYPE)

```
CREATE TYPE address (
street text,
city text,
zip_code int,
phones set<text>
 );

CREATE TYPE full_name (
first_name text,
last_name text
);
```

ALTER TABLE VIDEOS (SET)

Add a column tags (that can contain multiple tag values)

```
ALTER TABLE videos ADD tags SET<TEXT>;
INSERT INTO videos (..., tags) VALUES (..., {'tag1', 'tag2'});
UPDATE videos SET tags = tags + {'tag3'} WHERE id = 1 ;
UPDATE videos SET tags = tags - {'tag1'} WHERE id = 1 ;

DELETE tags FROM videos WHERE id= 1
```

ALTER TABLE VIDEOS (LIST)

- Add a column artists (that can contain multiple tag values)

```
ALTER TABLE videos ADD artists LIST<TEXT>;
INSERT INTO videos (..., artists) VALUES (..., ['A1', 'A2']);
UPDATE videos SET artists[1] = ['A3'] WHERE id = 1 ;
DELETE artists[0] FROM videos WHERE id= 1 ;
```

ALTER TABLE VIDEOS (MAP)

- Add a column realisateurs (that can contain multiple tag values)

ALTER TABLE videos ADD realisateurs MAP<TEXT, TEXT>;

UPDATE videos SET realisateurs = {‘nom’:’Dupont’} WHERE id = 1 ;

UPDATE videos SET realisateurs = realisateurs+ {‘prenom’:’Jean’} WHERE id = 1 ;

UPDATE videos SET realisateurs[‘nom’] = ‘machin’ WHERE id = 1

UDT

Create a video_encoding UDT following the example:

{encoding: '1080p', height: 1080, width: 1920, bit_rates: {'3000 Kbps', '4500 Kbps', '6000 Kbps'}

Field Name	Data Type
encoding	text
height	int
width	int
bit_rates	set<text>

Create a video_encoding.csv file containing video_id and the encoding information Example:

1,"{encoding: '1080p', height: 1080, width: 1920, bit_rates: {'3000 Kbps', '4500 Kbps', '6000 Kbps'}}"

ALTER TABLE AND ADD INFO

- Add a new encoding column to the videos table
- Insert new information from previously created videos_encoding.csv
- Show videos content

COUNTER

Create a new table with a counter to update the number of videos for each tag and year

```
CREATE TABLE videos_count_by_tag (
tag TEXT,
added_year INT,
video_count counter,
PRIMARY KEY (tag, added_year)
);
```

COUNTER

To update the counter: (launch some updates on the table)

```
UPDATE videos_count_by_tag SET video_count = video_count + 1 WHERE tag='MyTag' AND added_year=2015;
```

Display the result

Try a counter update with a tag and a year that does not exist in your table. What do you get?

TEMPORAL DATA

- Any value is associated with a TIMESTAMP.
- Automatic stamping (ms) during the update

```
CREAT TABLE user (id int primary key, name text);
INSERT INTO user (id, name) values (1, 'user 1');
INSERT INTO user (id, name) values (2, 'user 2');
INSERT INTO user (id, name) values (3, 'user 3');
select * from user;
select id, name, writetime(name) from user;
```

- Possible to specify with the query:

```
UPDATE user USING TIMESTAMP 12345 set name = 'user 4'
where id = 2; select id, name, writetime(name) from user;
UPDATE user USING TIMESTAMP 12344 set name = 'user 5'
where id = 2; select id, name, writetime(name) from user;
```

- We can delete a value defined at a time T:

DELETE name USING TIMESTAMP 12345 FROM user WHERE id=2;

This value can not be used in WHERE clause.

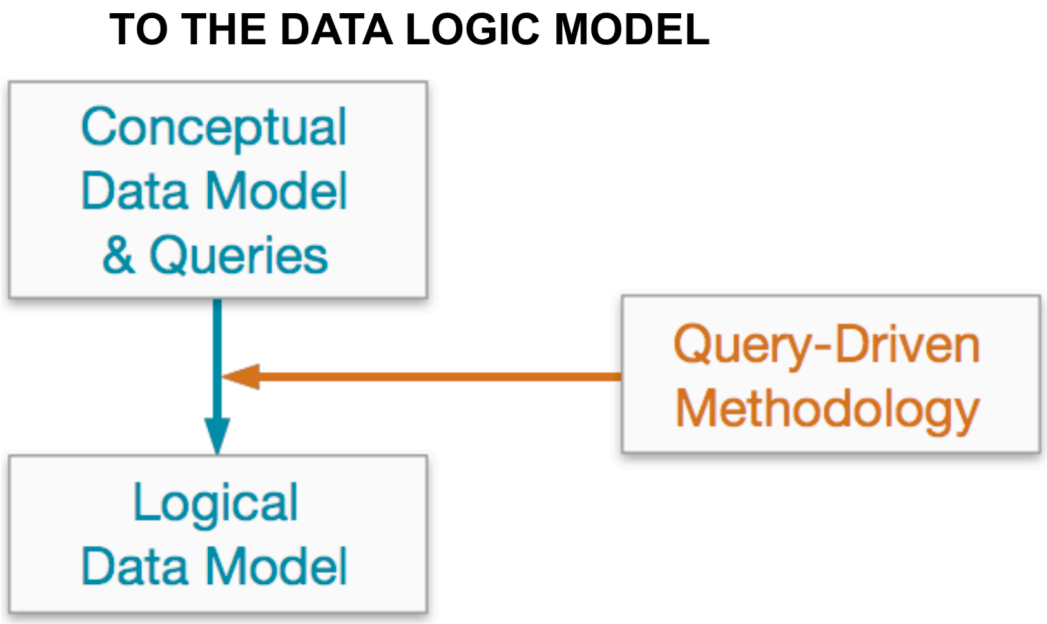
- Possible to manage volatile data: TTL

Same as TIMESTAMP, gives the number of seconds when the value is visible.

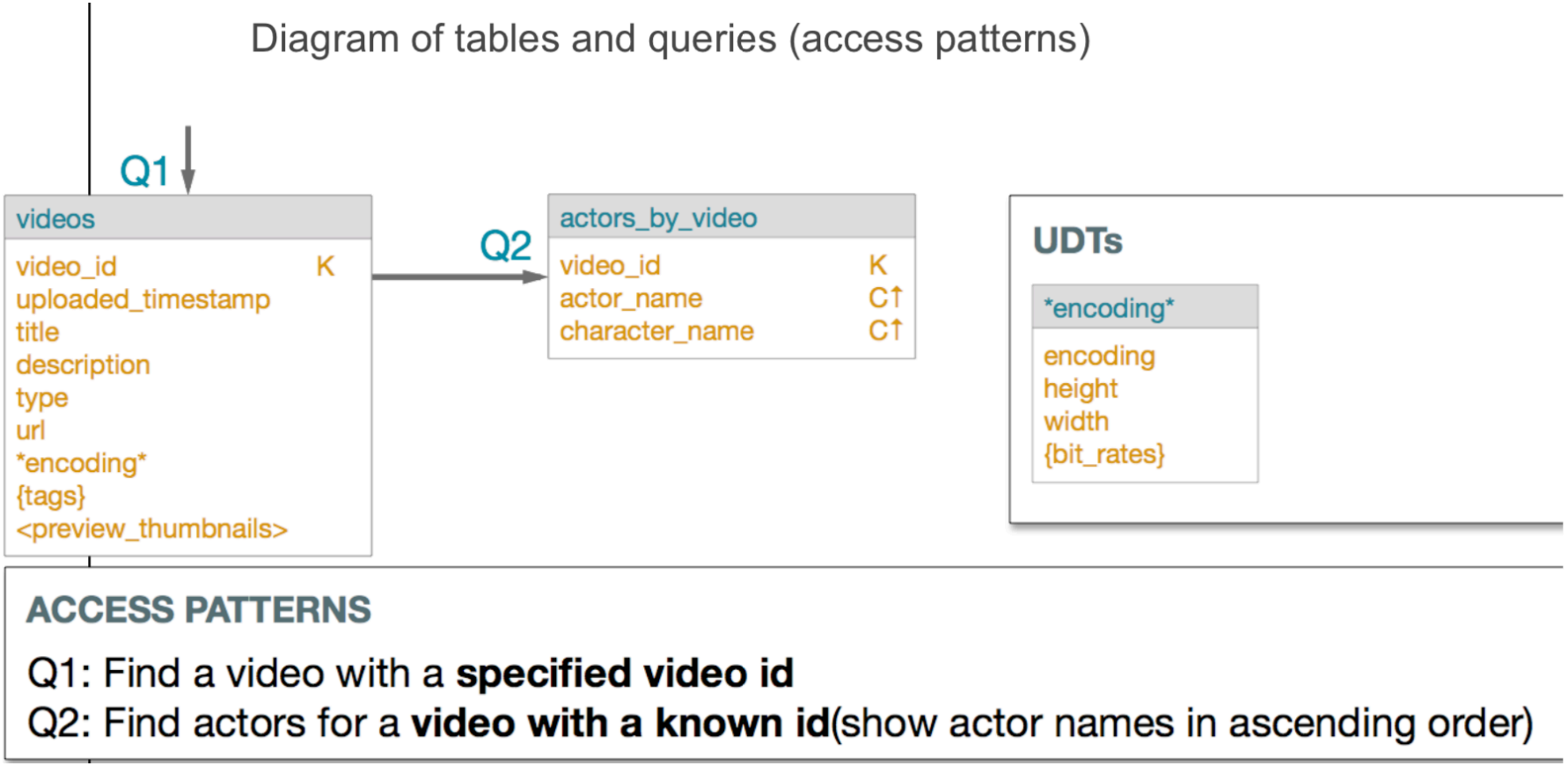
UPDATE user USING TTL 60 SET name = 'user 10' where id = 2;

select id, name, ttl(name) from user;

CONCEPTUAL DATA MODEL (CDM)



LOGIC MODEL: CHEBOTKO DIAGRAM



CHEBOTKO DIAGRAM: NOTATION