CASSANDRA – INTRODUCTION AND PRACTICAL LAB

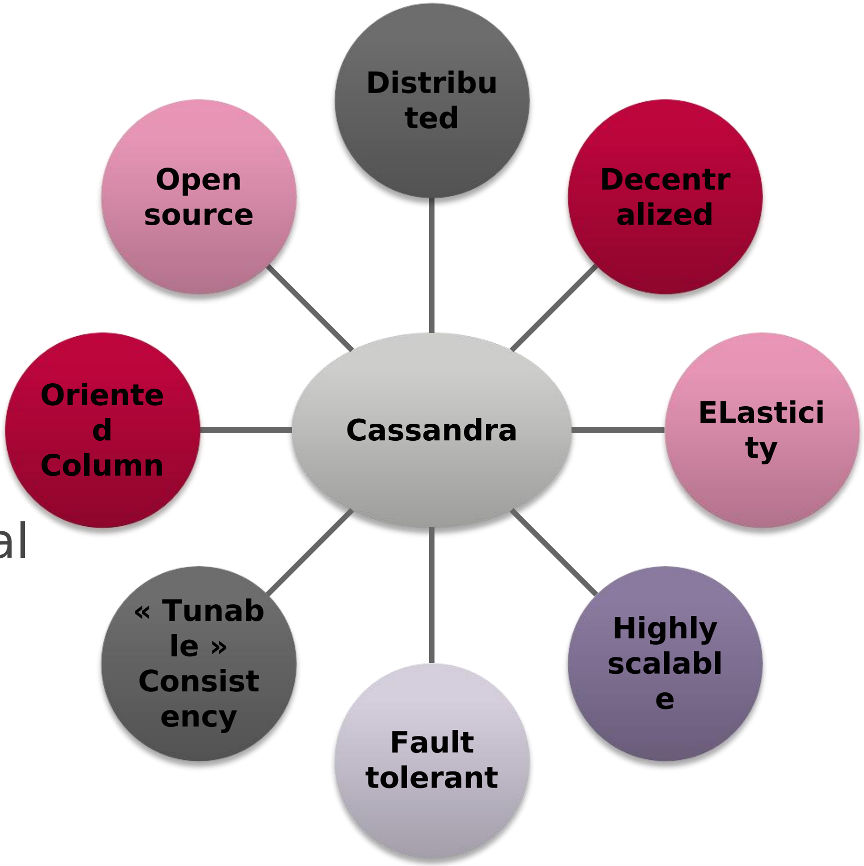
Rafael Angarita
ISEP – Paris
rafael.angarita@isep.fr

CASSANDRA

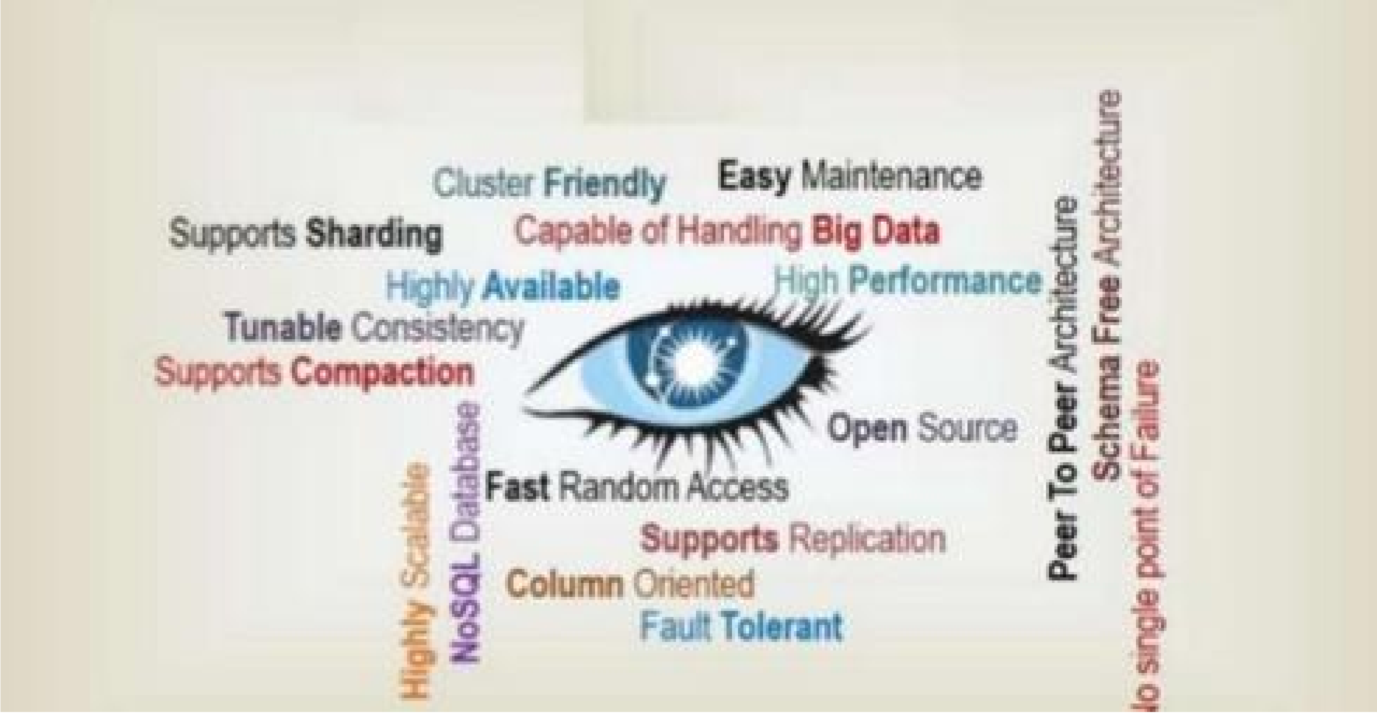
- Initiator: Facebook in 2007 (solution for the problem "inbox search »)
- Apache 2.0 License
- Written in Java
- Data Model
  - Based on BigTable (data model) and Dynamo (partitioning and consistency)
- Thrift interface: Ruby, Perl, Python, Scala and Java, ...
- CQL (Cassandra Query Language): SQL-Like

FONCTIONNALITÉS

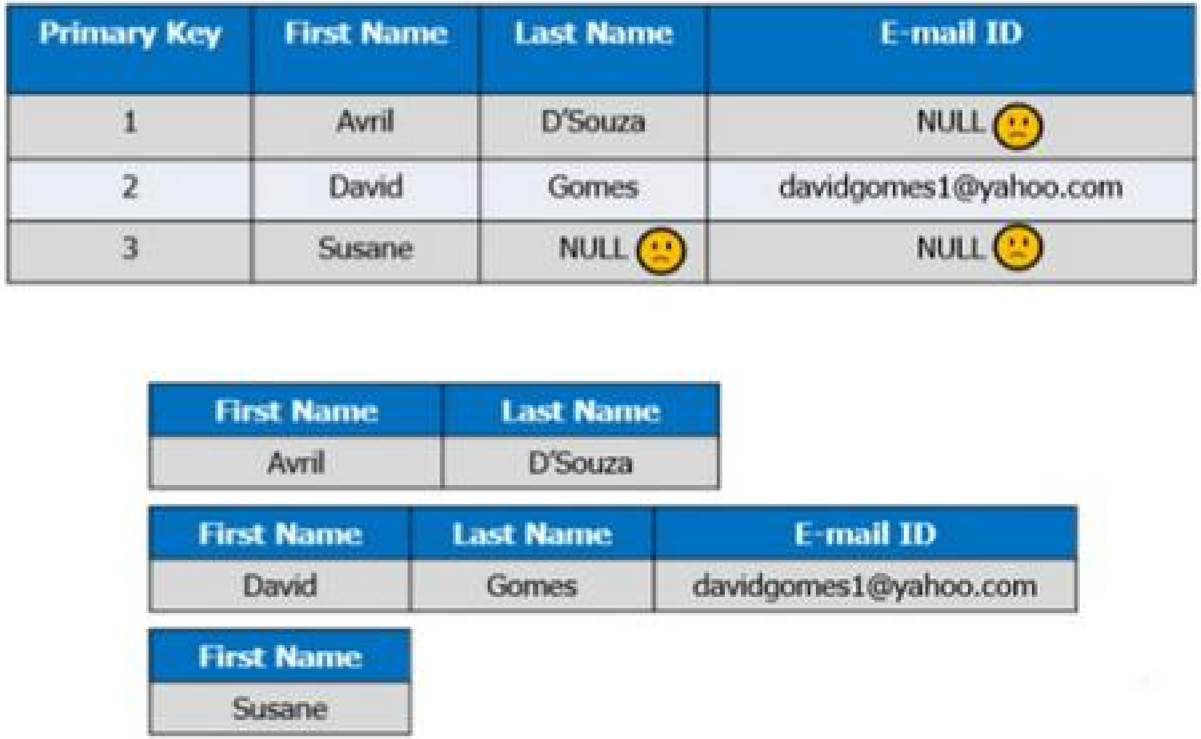
- Designed to handle large amount of data across multiple servers
- Easy to implement and deploy
- Mimics traditional relational database systems



CARACTÉRISTICS



SUITABLE FOR SPARSE DATA



Primary Key	First Name	Last Name	E-mail ID
1	Avril	D'Souza	NULL ☹
2	David	Gomes	davidgomes1@yahoo.com
3	Susane	NULL ☹	NULL ☹

First Name	Last Name
Avril	D'Souza

First Name	Last Name	E-mail ID
David	Gomes	davidgomes1@yahoo.com

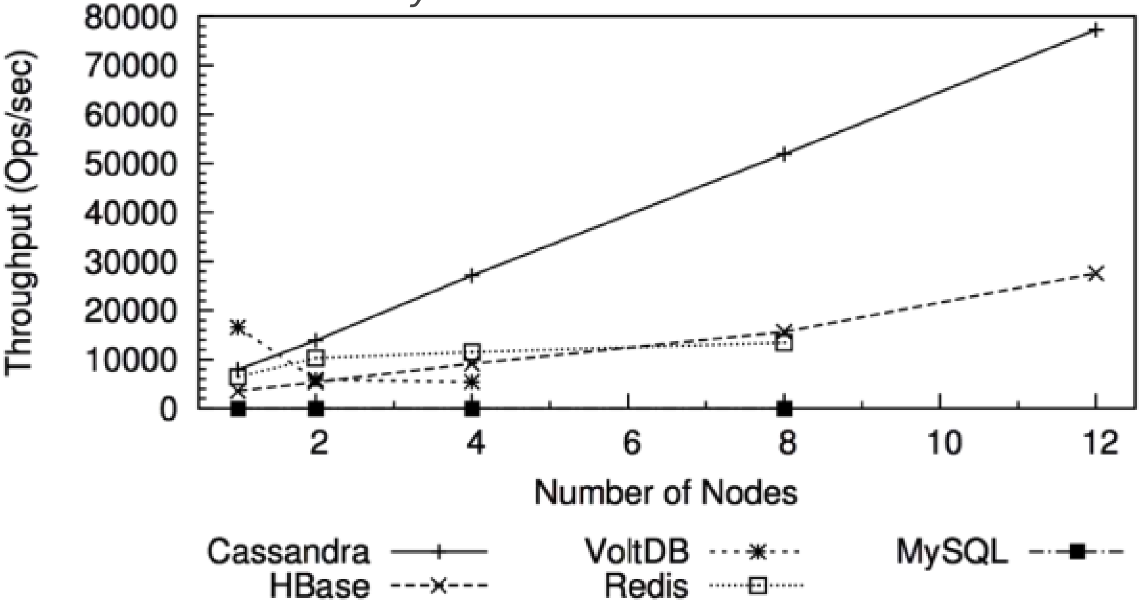
First Name
Susane

PERFORMANCE

Its design allows it to surpass competing DBs

Very good read / write rates: Improves linearly with the addition of new nodes

“In terms of scalability, there is a clear winner throughout our experiments. Cassandra achieves the highest throughput for the maximum number of nodes…” - University of Toronto



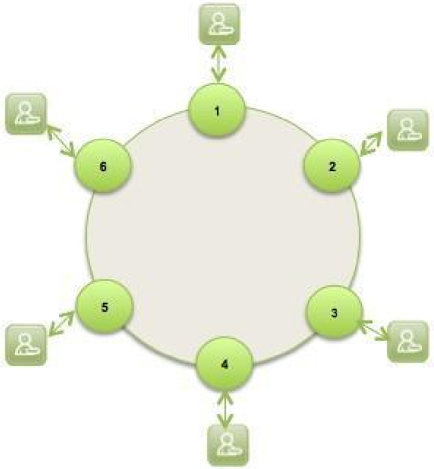
.com People, Businesses & Places
Houghton Mifflin Harcourt
Morningstar
platform46
accenture
adform
adku
Adobe
AdXpose
Agentis
aiMatch ad intelligence
Apixio
applifier
AppScale
appsavvy
Apptegic
Arkivum
Assistly
backupify
BrightEdge
Bronto
Burt.
CallFire
Catalyst SGC
Cisco
citypath
clearspring
cloudkick
CloudTalk
clover
Constant Contact
DachisGroup
Datadog
Datagraph, Inc.
Demandforce
despegar.com
digg
Digital Reasoning
Digital River
dotomi
druva
ebay
ebuddy
egnyte
embedly
Engine Yard
enkata
Ericsson
Expedia
FanBridge
Fight my Monster
Finn
formspring
Frugal Mechanic
Gemini
GigaSpaces
GoTime
Gowalla
GridPoint
Healthline
Health Market Science
HealthX
hobsons
hp
HubSpot
IBM
Impetus
inkling
iovation
Isidorey
jetpac
Jumptap
junction networks
justSignal
librato
LiveOffice
LivePerson
Mahalo
martinimedia
media6degrees
Medialets
MedProcure
Mollom
monitis
movile
napshare
Navteq
Netflix
Nextag
Next Big Sound
NextSlide
ngmoco:)
nicira
ONEsite
one spot
Ooyala
OpenAdExchange
OpenFeint
opentracker
Openwave
Other inbox
outbrain
page2rss
Palantir
Pantheon
Pitney Bowes
PBS Kids
PCPhase
PeopleBrowsr
plaxo
Polyvore
PostRank
Press+
proofpoint
quantiFind
Quova
rackspace
Rakuten
Rally
ReachLocal
real
reddit
referentia
retickr
Rhapsody
RockMelt
rockyou
Ruckus Wireless
Scandit
SEOmoz
Seven
Shazam
shopsavvy
SimpleGeo
skillpages
smeet
SocialFlow
SPI
Sonico
SoundCloud
spidertracks
Spotify
Spredfast
springpad
Squidoo
Symantec
Tapulous
Teads
Tellybug
Tendril
thefind
trap!t
TwImpact
twitter
Urban Airship
usergrid
VigLink
Visual DNA
VTEX
wajam
weR
where
Wildfire
WSO2
Wupa
xobni
Yakaz
YCharts
ZipZapPlay
zoominfo

2012

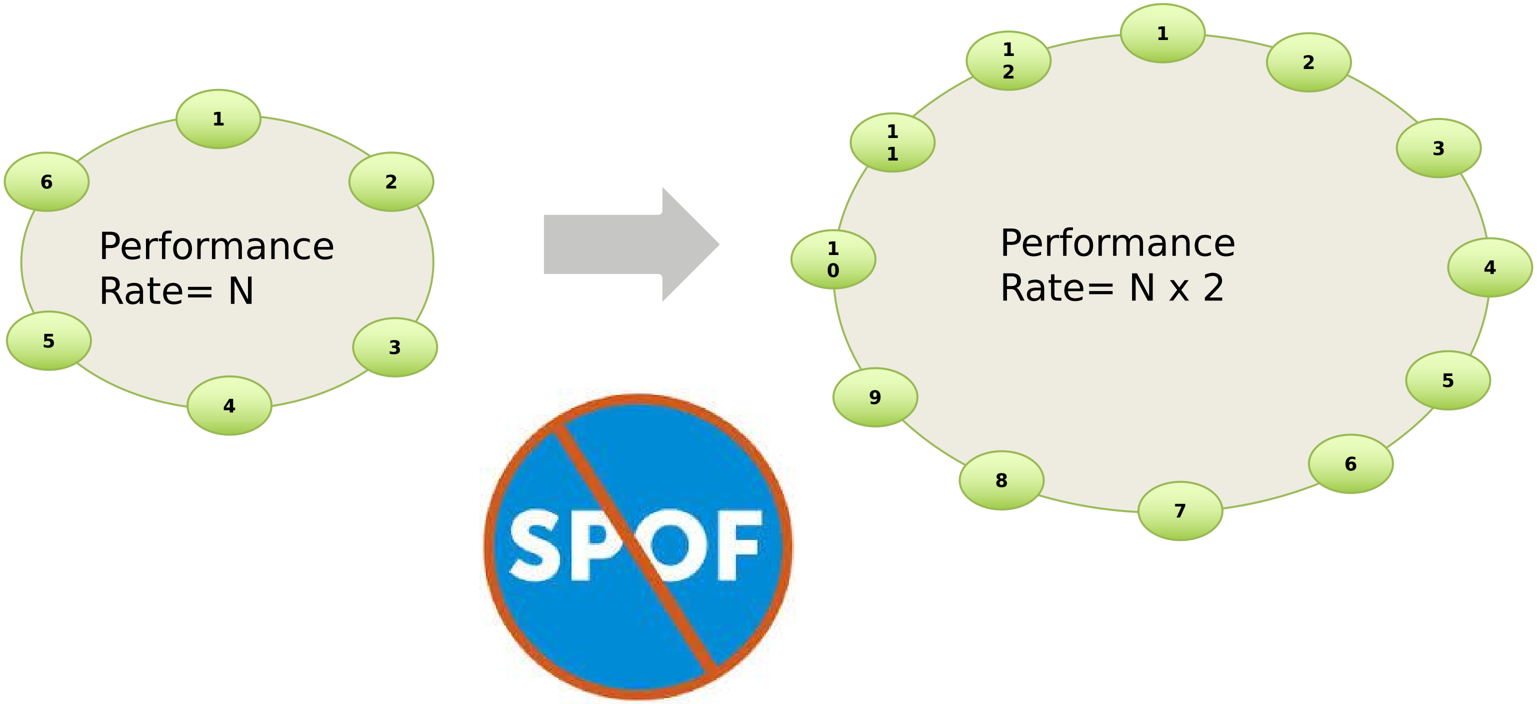
CASSANDRA ARCHITECTURE

OVERVIEW

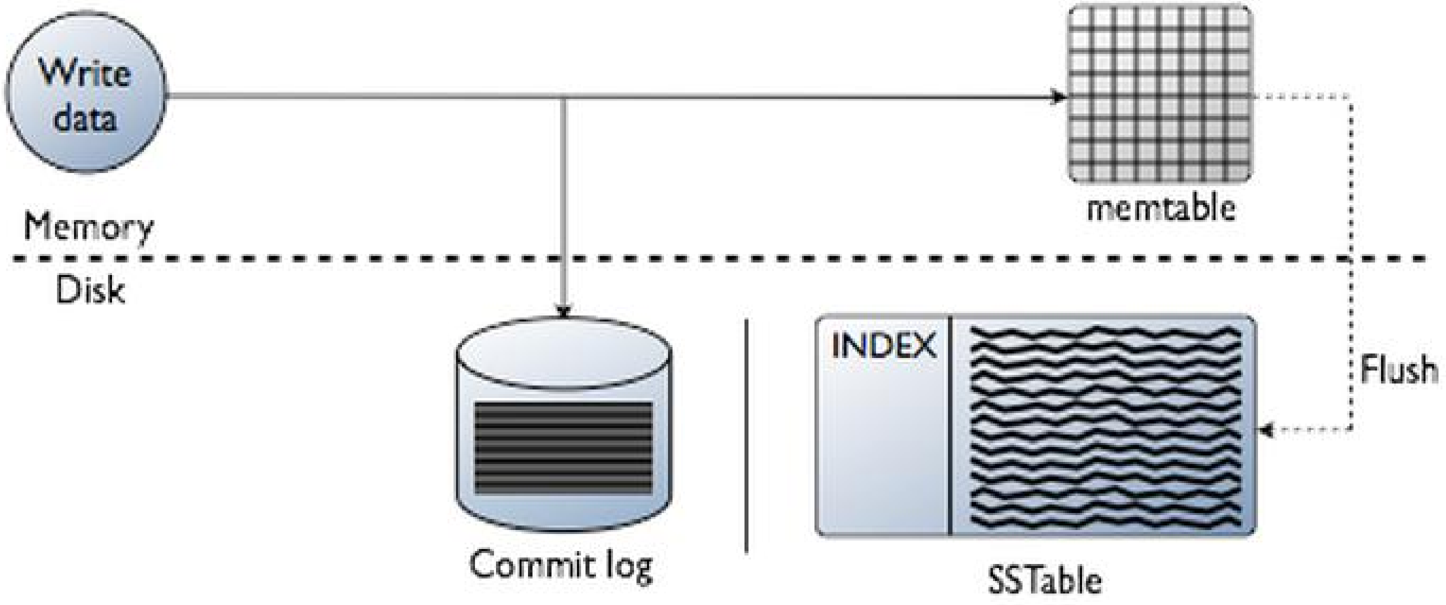
- Cassandra was designed with the understanding that system/ hardware failures can and do occur
- Peer-to-peer, distributed system
- All nodes are the same
- Data partitioned among all nodes in the cluster
- Custom data replication to ensure fault tolerance
- Read/Write-anywhere design
- Google BigTable - data model
  - Column Families
  - Memtables
  - SSTables
- Amazon Dynamo - distributed systems technologi
  - Consistent hashing
  - Partitioning
  - Replication
  - One-hop routing



SCALING OUT ET HIGH AVAILABILITY



WRITE OPERATIONS



WRITE OPERATIONS

- Commit log
  - First place a write is recorded
  - Crash recovery mechanism
- MemTable
  - Data structure in memory
  - Once recorded in commit log, data is written to Memtable
  - Once memtable size reaches a threshold, it is flushed (appended) to SSTable
  - First place read operations look for data
- SSTable
  - Kept on disk
  - Immutable once written
  - Periodically compacted for performance

CONSISTENCY IN CASSANDRA

- Architecture « **Read and Write Everywhere** »
- The user can connect to any node, any data center, and read / write the data he wants
- Cassandra the fastest NoSQL database in **write operations**
- Extension of the notion of eventual consistency to **a tunable consistency**
- The choice of the consistency is made by request: clause USING CONSISTENCY (the ONE level is by default)

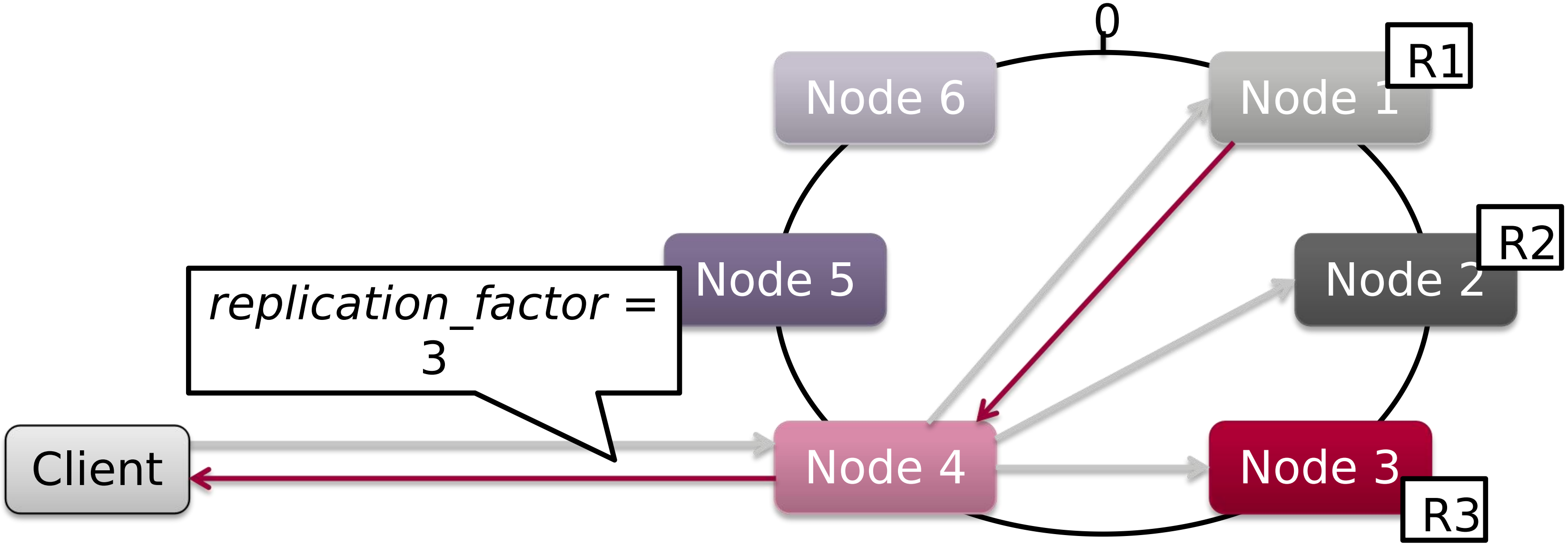
write consistency

Level	Description
ONE	1 replica.
QUORUM	(N /2) + 1
ALL	N = replication factor

read consistency

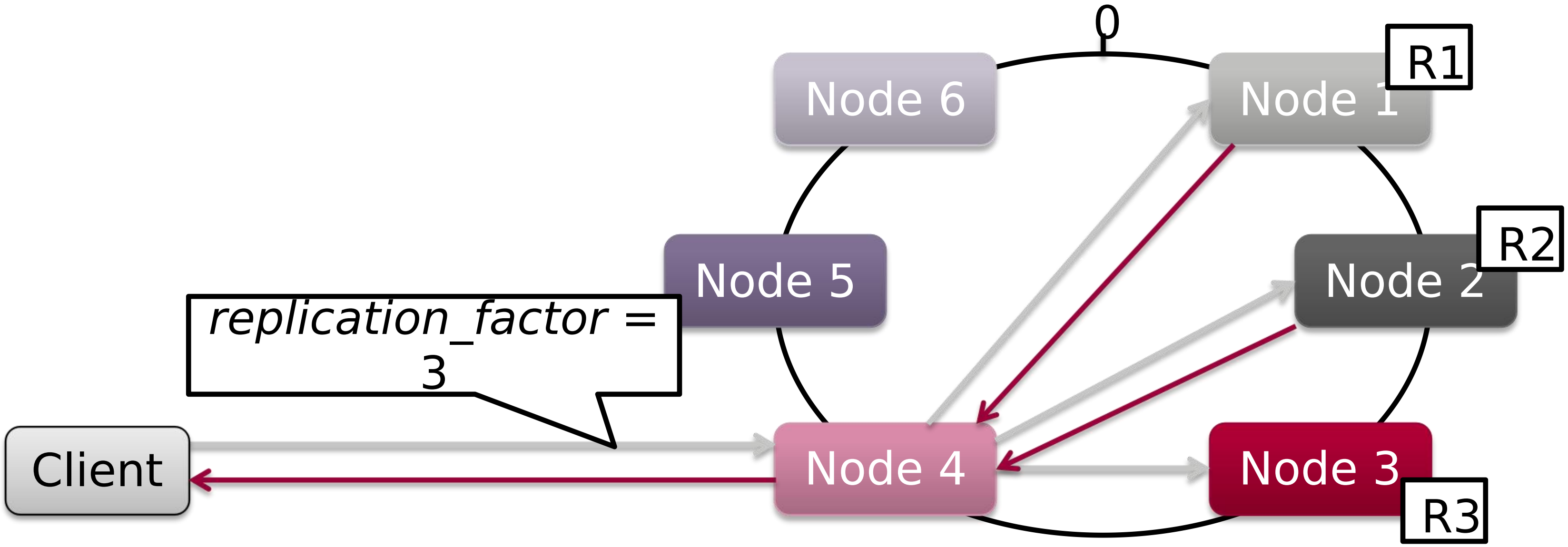
Level	Description
ONE	1 replica
QUORUM	Return most recent TS after (N /2) + 1 report
ALL	N = replication factor

WRITE- CONSISTENCY ONE



INSERT INTO table (column1, …) VALUES (value1, …) USING CONSISTENCY ONE

WRITE – CONSISTENCY QUORUM



INSERT INTO table (column1, …) VALUES (value1, …) USING CONSISTENCY QUORUM

GENERALITIES

- Cassandra is designed as a distributed database management system
  - use it when you have a lot of data spread across multiple servers
- Cassandra write performance is always excellent, but read performance depends on write patterns
  - it is important to spend enough time to design proper schema around the query pattern

STRENGTHS AND WEAKNESSES

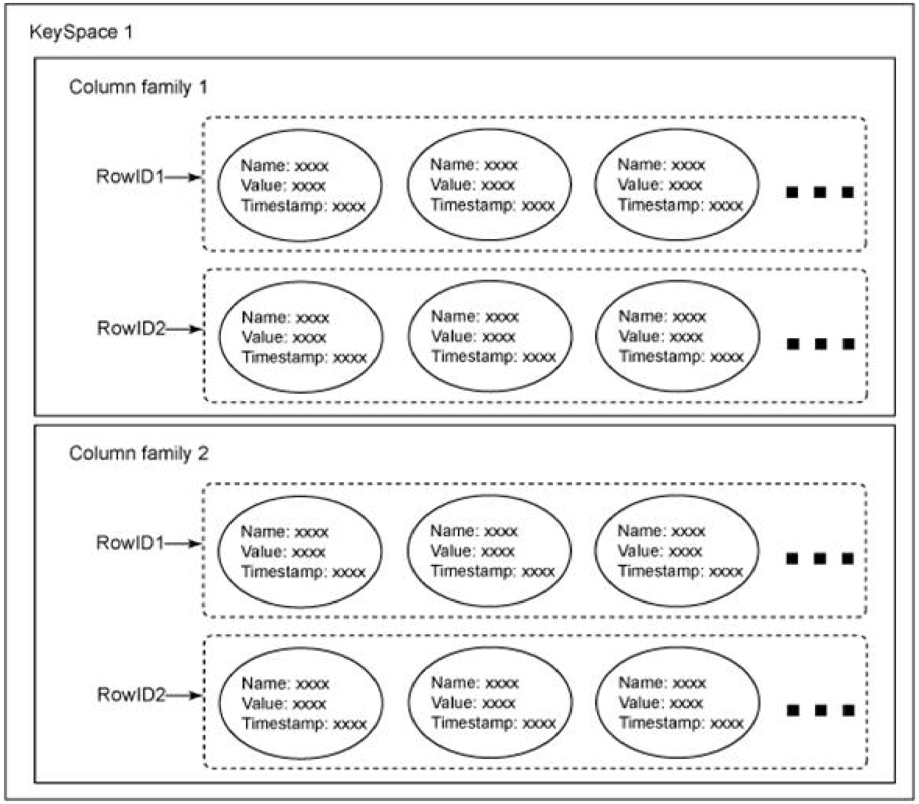
- perfect for time-series data
- high performance
- Decentralization
- nearly linear scalability
- replication support
- no single points of failure
- MapReduce support

- no referential integrity
- no concept of JOIN
- querying options for retrieving data are limited
- sorting data is a design decision
- no GROUP BY
- first think about queries, then about data model

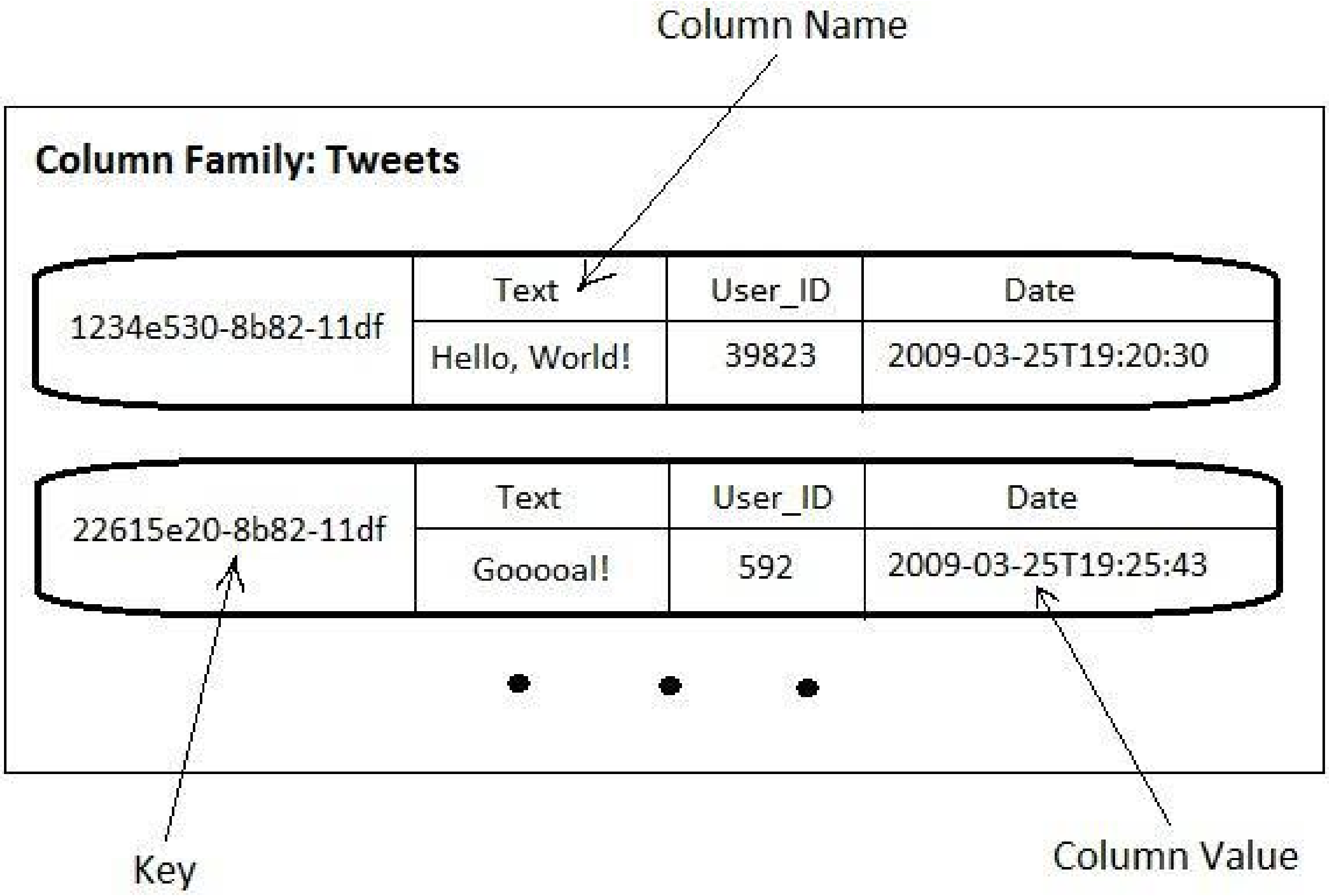
DATA MODEL

KEY-COLUMN(S) MODEL

- A record is a collection of labeled columns (with a name)
- Family of columns = Table (By analogy with RDBMS)
- A record must contain at least one column



EXAMPLE



Column Family: Tweets

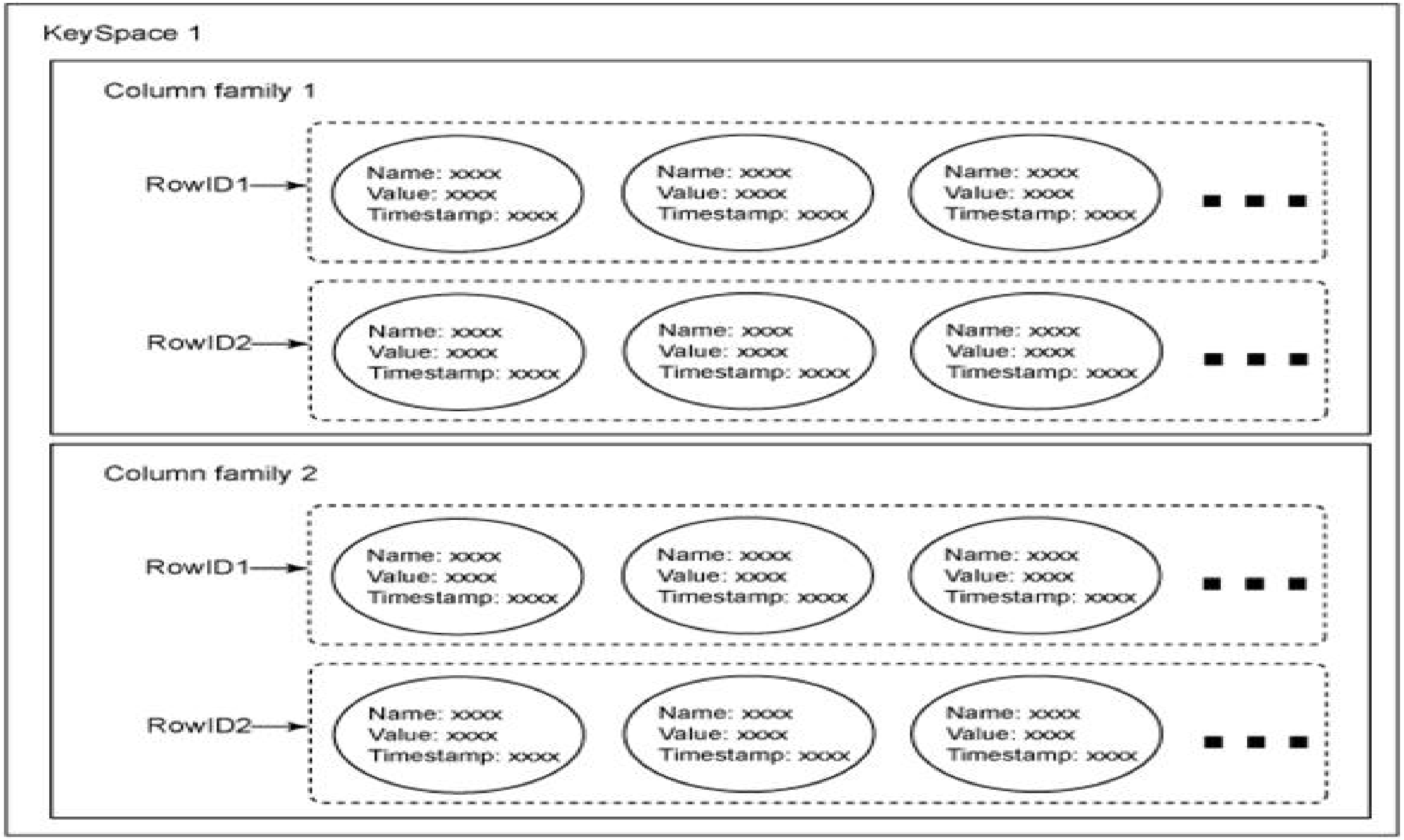
1234e530-8b82-11df	Text	User_ID	Date
	Hello, World!	39823	2009-03-25T19:20:30

22615e20-8b82-11df	Text	User_ID	Date
	Goooal!	592	2009-03-25T19:25:43

• • •

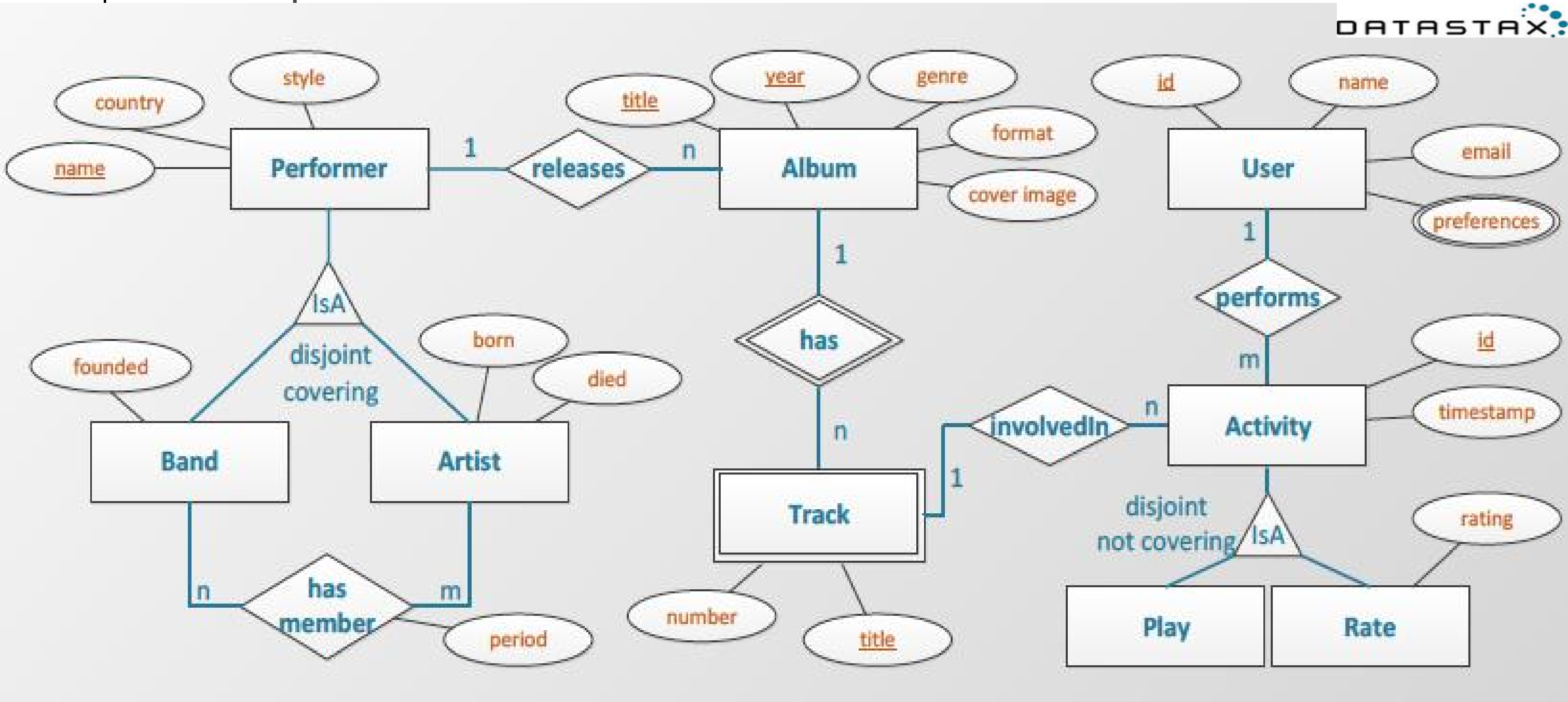
KEYSPACE

Set of column families (~ Database)

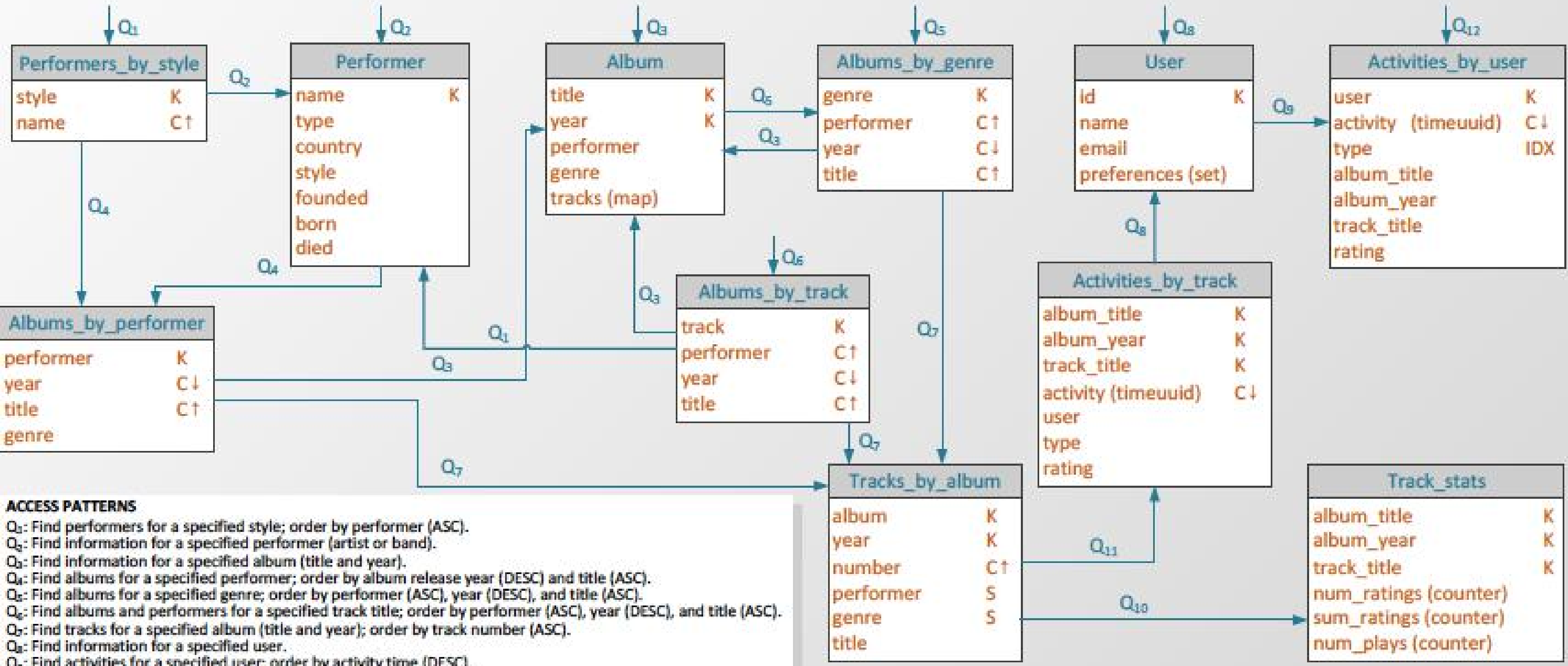


METHODOLOGY – E/R MODEL

Diagramm ER (Chen): entities, associations, cardinalities,



DATA MODEL FOR CASSANDRA



Performers_by_style	
style	K
name	C↑

Performer	
name	K
type	
country	
style	
founded	
born	
died	

Album	
title	K
year	K
performer	
genre	
tracks (map)	

Albums_by_genre	
genre	K
performer	C↑
year	C↓
title	C↑

User	
id	K
name	
email	
preferences (set)	

Activities_by_user	
user	K
activity (timeuuid)	C↓
type	IDX
album_title	
album_year	
track_title	
rating	

Albums_by_performer	
performer	K
year	C↓
title	C↑
genre	

Albums_by_track	
track	K
performer	C↑
year	C↓
title	C↑

Activities_by_track	
album_title	K
album_year	K
track_title	K
activity (timeuuid)	C↓
user	
type	
rating	

Tracks_by_album	
album	K
year	K
number	C↑
performer	S
genre	S
title	

Track_stats	
album_title	K
album_year	K
track_title	K
num_ratings (counter)	
sum_ratings (counter)	
num_plays (counter)	

ACCESS PATTERNS

Q_1: Find performers for a specified style; order by performer (ASC).
Q_2: Find information for a specified performer (artist or band).
Q_3: Find information for a specified album (title and year).
Q_4: Find albums for a specified performer; order by album release year (DESC) and title (ASC).
Q_5: Find albums for a specified genre; order by performer (ASC), year (DESC), and title (ASC).
Q_6: Find albums and performers for a specified track title; order by performer (ASC), year (DESC), and title (ASC).
Q_7: Find tracks for a specified album (title and year); order by track number (ASC).
Q_8: Find information for a specified user.
Q_9: Find activities for a specified user; order by activity time (DESC).
Q_{10}: Find statistics for a specified track.
Q_{11}: Find user activities for a specified track; order by activity time (DESC).
Q_{12}: Find user activities for a specified activity type.
...

CASSANDRA QUERY LANGUAGE - CQL

KEYSPACE

```
CREATE KEYSPACE demo
        WITH replication = {‘class’: ’SimpleStrategy’,
        replication_factor’: 3};

USE demo;

DROP KEYSPACE demo;
```

CREATING A TABLE (COLUMN FAMILY)

```
CREATE TABLE users(
 email varchar,
 bio varchar,
 birthday timestamp,
 active boolean,
 PRIMARY KEY (email));
```

```
CREATE TABLE tweets(
  email varchar PRIMARY KEY,
  time_posted timestamp,
  tweet varchar);
```

INSERTION

```
INSERT INTO users (email, bio, birthday, active)
          VALUES (‘isep.rdi@gmail.com’, ‘Associate
professor’,
                    516513600000, true);
```

Import data from csv file

```
COPY table1 (column1, column2, column3, column4)
FROM ’data.csv';
```

With header

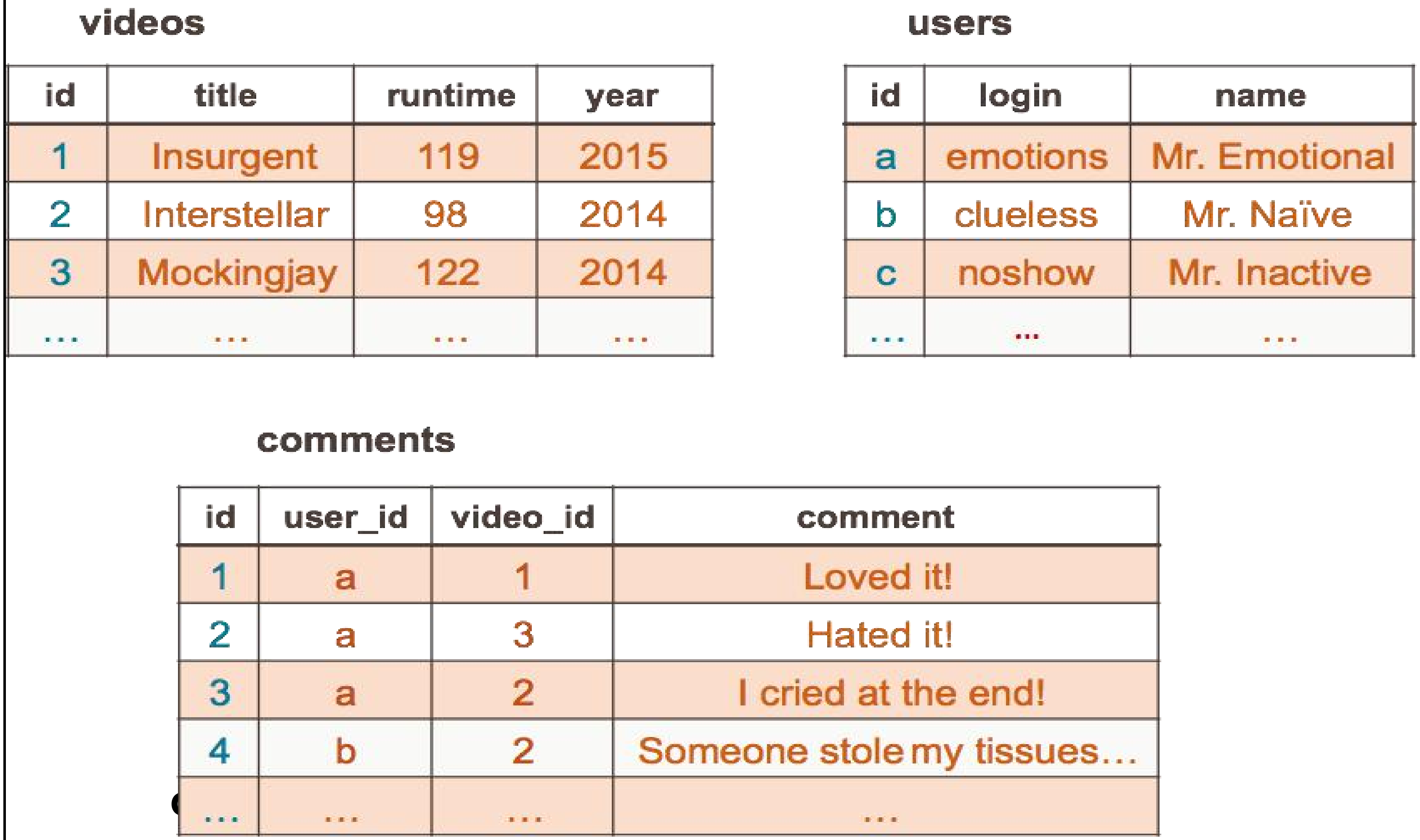
```
COPY table1 (column1, column2, column3, column4)
FROM ’data.csv’
WITH HEADER=true;
```

QUERYING

```
SELECT * FROM users;

SELECT count(*) from users;

SELECT * FROM users LIMIT 10;

SELECT email FROM users WHERE active = true;
```

DENORMALISATION

REMINDERS - EXAMPLE



videos

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

users

id	login	name
a	emotions	Mr. Emotional
b	clueless	Mr. Naïve
c	noshow	Mr. Inactive
…	…	…

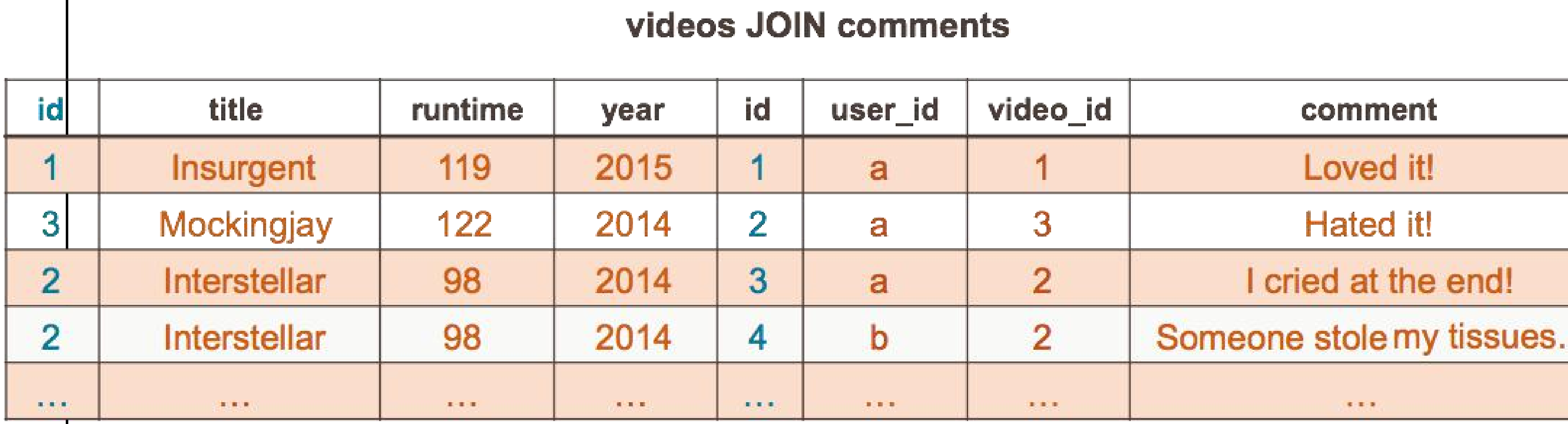
comments

id	user_id	video_id	comment
1	a	1	Loved it!
2	a	3	Hated it!
3	a	2	I cried at the end!
4	b	2	Someone stole my tissues…
…	…	…	…

COMMENTS ON EACH VIDEO

SELECT comment

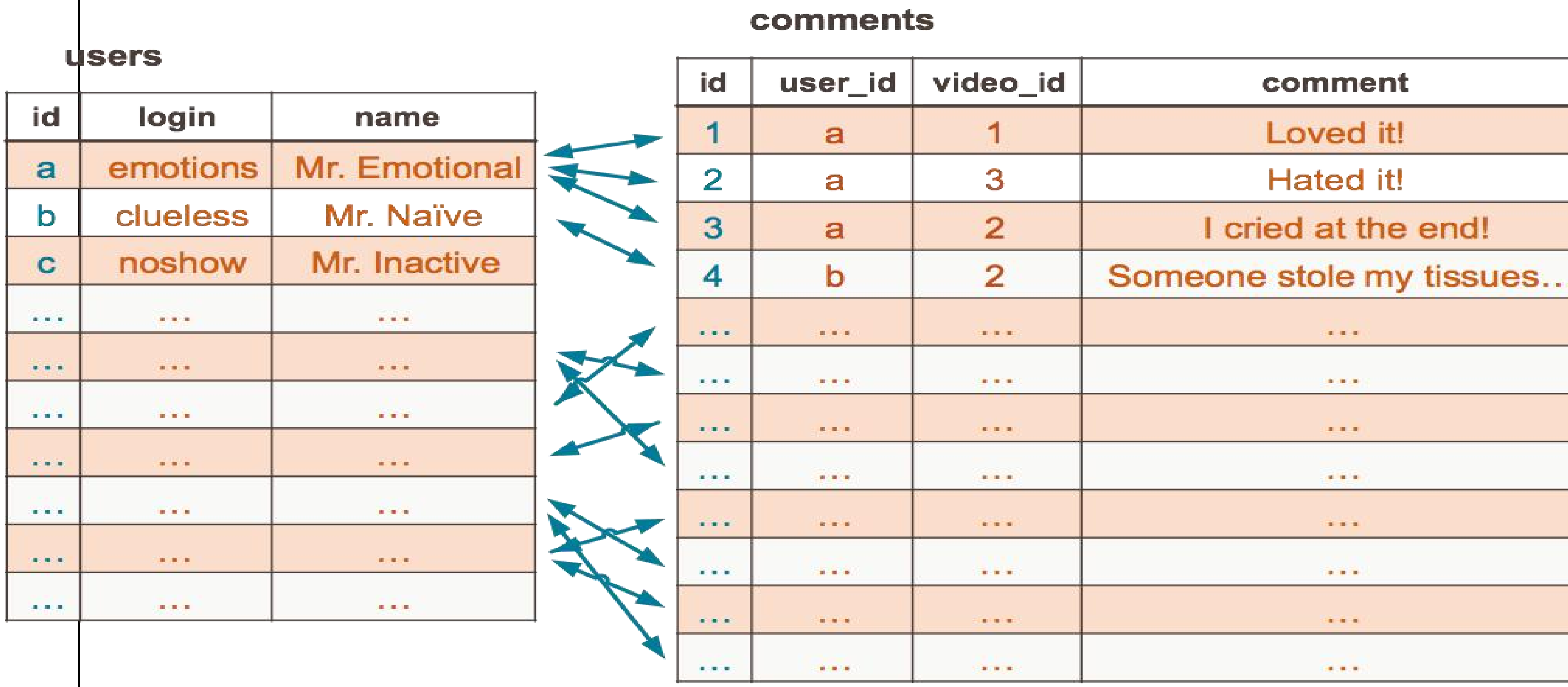
FROM videos JOIN comments ON videos.id = comments.video_id



videos JOIN comments

id	title	runtime	year	id	user_id	video_id	comment
1	Insurgent	119	2015	1	a	1	Loved it!
3	Mockingjay	122	2014	2	a	3	Hated it!
2	Interstellar	98	2014	3	a	2	I cried at the end!
2	Interstellar	98	2014	4	b	2	Someone stole my tissues.
...	...	...	...	...	...	...	...

COMMENTS ON EACH LOGIN AND USER



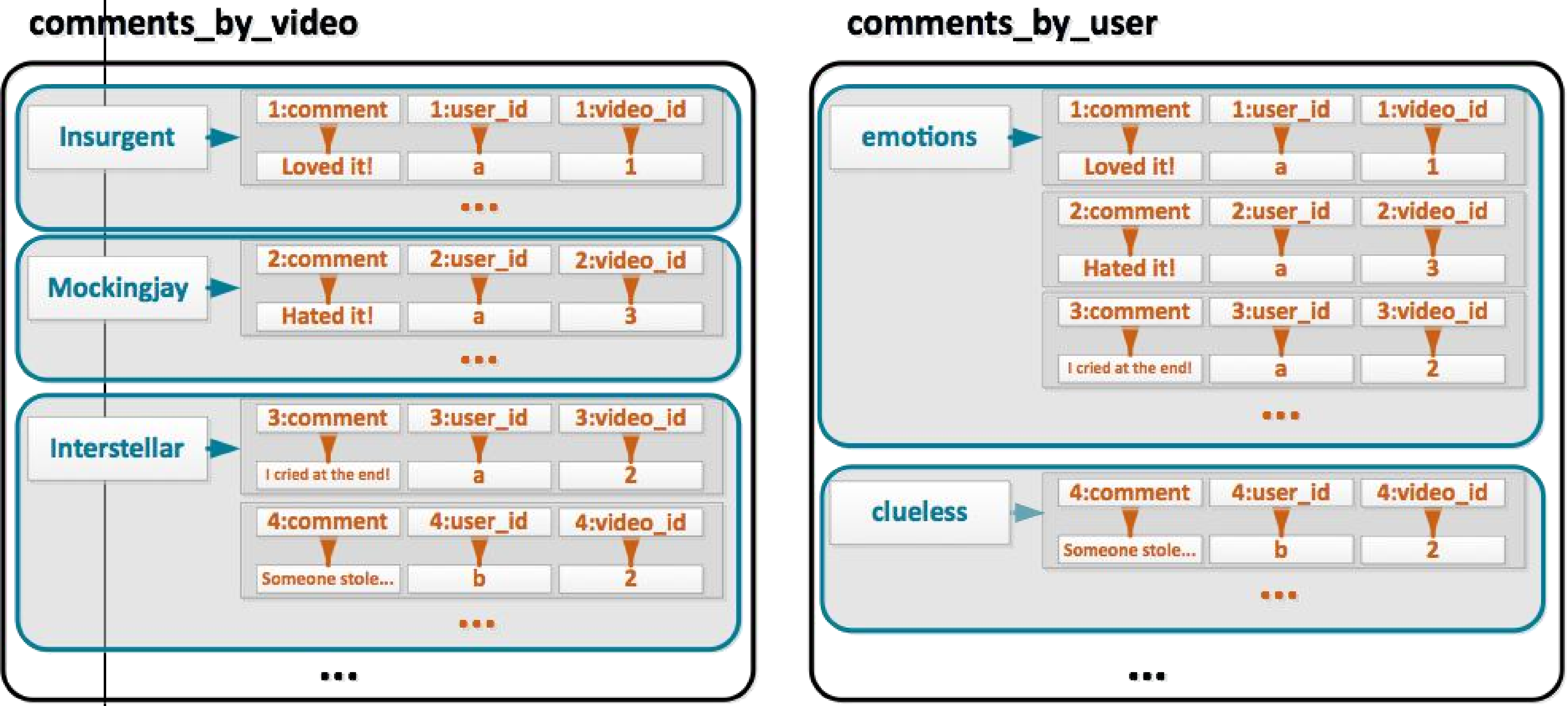
users

id	login	name
a	emotions	Mr. Emotional
b	clueless	Mr. Naïve
c	noshow	Mr. Inactive
...	...	...
...	...	...
...	...	...
...	...	...
...	...	...
...	...	...
...	...	...

comments

id	user_id	video_id	comment
1	a	1	Loved it!
2	a	3	Hated it!
3	a	2	I cried at the end!
4	b	2	Someone stole my tissues..
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...

DENORMALISATION IN CASSANDRA



comments_by_video

Insurgent
1:comment → Loved it!
1:user_id → a
1:video_id → 1
...

Mockingjay
2:comment → Hated it!
2:user_id → a
2:video_id → 3
...

Interstellar
3:comment → I cried at the end!
3:user_id → a
3:video_id → 2
4:comment → Someone stole...
4:user_id → b
4:video_id → 2
...

...

comments_by_user

emotions
1:comment → Loved it!
1:user_id → a
1:video_id → 1
2:comment → Hated it!
2:user_id → a
2:video_id → 3
3:comment → I cried at the end!
3:user_id → a
3:video_id → 2
...

clueless
4:comment → Someone stole...
4:user_id → b
4:video_id → 2
...

...

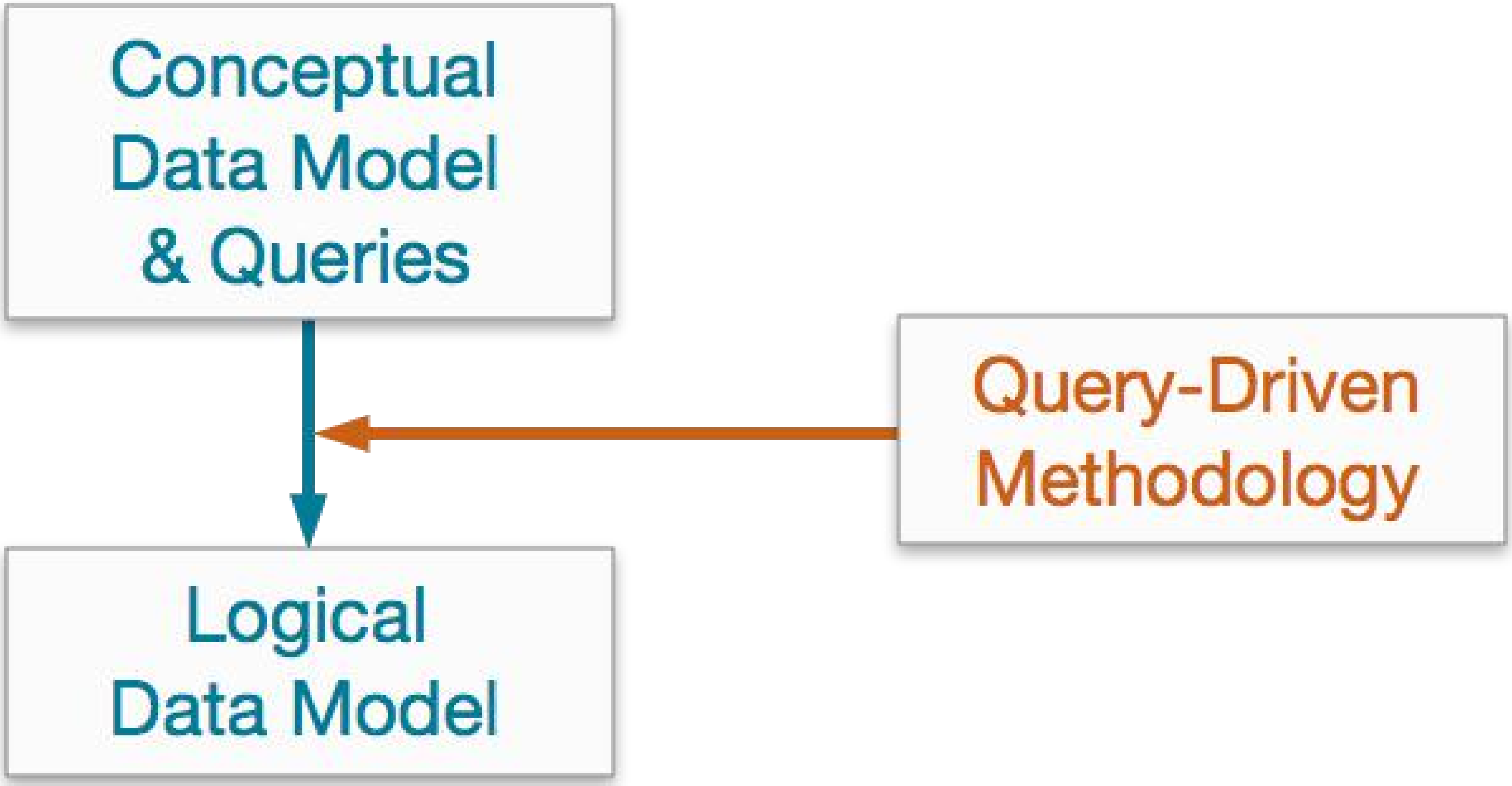
TABLES

Create tables and insert records
Write queries to find comments for a particular movie / user

We want to classify the videos commented by a user from the most recent to the oldest. What do we need to do?

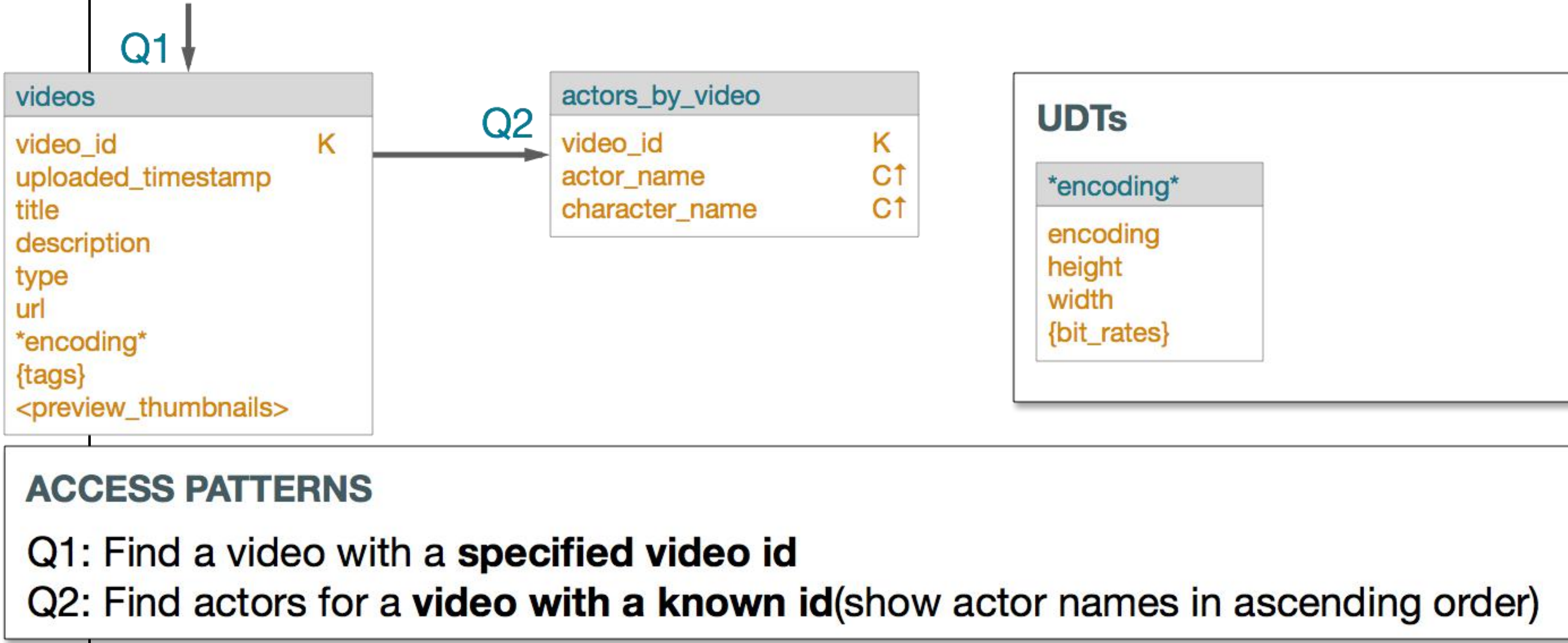
CONCEPTUAL DATA MODEL (CDM)

TO THE DATA LOGIC MODEL



LOGIC MODEL: CHEBOTKO DIAGRAM

Diagram of tables and queries (access patterns)



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↑

UDTs

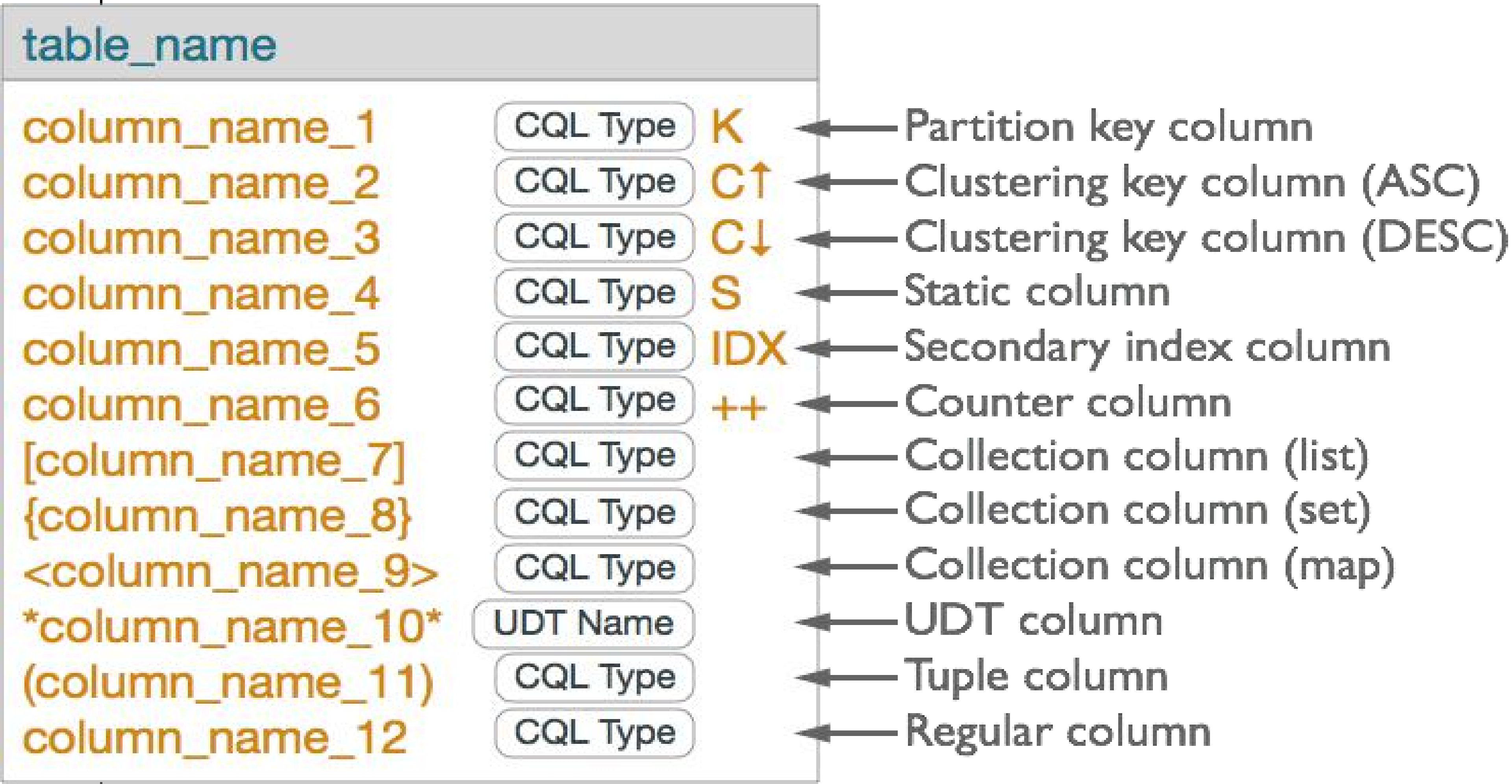
encoding
encoding
height
width
{bit_rates}

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: NOTATION



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

CASSANDRA-CONCLUSION

perfect for time-series data

high performance

Decentralization

nearly linear scalability

replication support

no single points of failure

MapReduce support

first think about queries, then about data model

MORE INFORMATION

Dev: http://www.datastax.com/dev
Docs:http://docs.datastax.com/en/index.html
Planet Cassandra: http://planetcassandra.org/
blogs: http://tobert.github.io/
http://patrickmcfadin.com/
http://rustyrazorblade.com/
https://ahappyknockoutmouse.wordpress.com/author/anukeus/
http://thelastpickle.com/blog/
Livre : http://www.amazon.com/Cassandra-High-Availability-Robbie-Strickland/dp/1783989122
DataStax Academy: https://academy.datastax.com/
Formation: http://www.datastax.com/what-weoffer/products-services/training