

# Apache Cassandra

---

A WIDE-COLUMN NOSQL DATABASE

# Getting started

---

# Introduzione

---

Cassandra è un database NoSQL wide-column, ispirato al modello BigTable

I database wide-column riprendono il concetto relazionale di *riga*, ma lo modificano sostanzialmente

- Una riga è sempre associata ad una chiave (primaria)
- Ogni riga contiene la propria definizione di colonne e valori

Alcune delle caratteristiche principali:

- Gestione decentralizzata di repliche e scritture (**peer-to-peer**)
  - Garantisce un'elevata disponibilità del sistema e non presenta un *single point of failure*
- Scalabilità semplice e lineare
- Il livello di consistenza (one, quorum, all) è settato in fase di interrogazione
- **CQL** (Cassandra Query Language)

# Installazione

---

## Installazione di base

- Cassandra è avviato all'interno di un container Docker
- Per accedere al container eseguire il comando
  - `docker exec -it cassandra /bin/bash`
- Avviare la shell CQL
  - `cqlsh`

DataStax è il principale fornitore di software basati su Cassandra

- **DataStax Enterprise (DSE):** gratuito per scopi non commerciali
- **DataStax Studio e DevCenter:** strumenti per interrogare i dati

# [Obsoleto] Installazione

---

## Installazione di base

- Scaricare l'ultima versione dal sito: <http://cassandra.apache.org/download/>
- Scompattare l'archivio ed impostare la variabile d'ambiente JAVA\_HOME
- Avviare Cassandra: bin/cassandra -f -R
- Avviare la shell CQL: cqlsh

DataStax è il principale fornitore di software basati su Cassandra

- DataStax Community Edition: gratuito, non più supportato (obsoleto)
- DataStax Enterprise (DSE): gratuito per scopi non commerciali
- DataStax OpsCenter: strumento per configurare e gestire un cluster
- DataStax Studio e DevCenter: strumenti per interrogare i dati

# [Obsoleto] Installazione

---

Il sito di DataStax propone una sezione Academy per imparare ad utilizzare Cassandra e la suite software DataStax

- <https://academy.datastax.com/>
- Necessaria la registrazione (gratuita)

## Video-tutorial

- <https://academy.datastax.com/courses>

## DataStax Sandbox

- Una macchina virtuale Linux già configurata per lavorare subito con Cassandra
- [https://portal.datastax.com/downloads.php?dsedownload=tar/enterprise/sandbox/DataStax\\_Sandbox.ova](https://portal.datastax.com/downloads.php?dsedownload=tar/enterprise/sandbox/DataStax_Sandbox.ova)

# [Obsoleto] Installazione

---

Fonte di riferimento: <https://academy.datastax.com/resources/ds220-data-modeling>

Macchina virtuale:

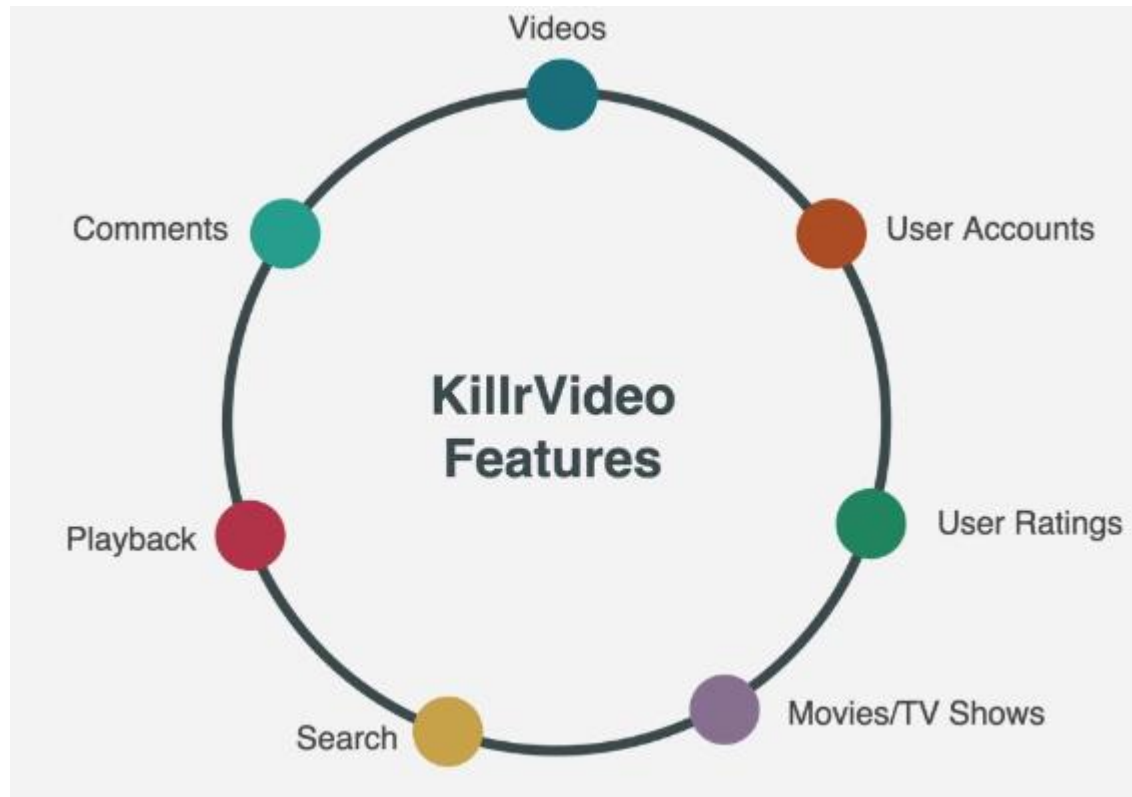
<https://s3.amazonaws.com/datastaxtraining/VM/DS220-vm-Jul2015.zip>

- Più leggera di quella indicata nella slide precedente
- Contiene file di dati da usare per le esercitazioni
- (attenzione: usa una versione di Cassandra del 2015)

# KillrVideo dataset

---

Database di esempio, che supporta un sito per la condivisione di video





# DDL

Keyspace = Database

- Class = strategia di sharding

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

USE killrvideo;
```

Table = Column family

```
CREATE TABLE users (
  user_id UUID,
  first_name TEXT,
  last_name TEXT,
  PRIMARY KEY (user_id)
);
```



Users		
user	email	name
a7e78478-0a54-4949-90f3-14ec4cbea40c	jbellis@datastax.com	Jonathan
67657da3-4443-46ab-b60a-510a658fc7bb	matt@datastax.com	Matt
3b1f62b1-386b-46e3-b55d-00f1abbafb2b	patrick@datastax.com	Patrick

# Tipi di dato

---

Text, Int, ...

UUID (Universally Unique Identifier)

- Generato chiamando *uuid()*
- Es: 52b11d6d-16e2-4ee2-b2a9-5ef1e9589328

TimeUUID

- UUID + timestamp del momento di insert
- Tipicamente utilizzato per memorizzare *time series*
- Es: 1be43390-9fe4-11e3-8d05-425861b86ab6

# Import di dati

---

Il comando copy permette di caricare dati da file esterni

- Header = true → salta la prima riga
- Le colonne si possono omettere se coincidono con quelle della tabella

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

# Select

---

Molto simile ad SQL

```
SELECT *  
FROM table1;
```

```
SELECT column1, column2, column3  
FROM table1;
```

```
SELECT COUNT(*)  
FROM table1;
```

```
SELECT *  
FROM table1  
LIMIT 10;
```

# Esercizio 1

---

## Setup del keyspace killrvideo

- Creare il keyspace
- Creare una tabella VIDEOS per memorizzare i video con le colonne indicate
- Caricare dati nella tabella
  - `/root/labwork/exercise-2/videos.csv`
- Visualizzare i dati con una SELECT
- Familiarizzare con LIMIT e COUNT(\*)

Column Name	Data Type
<u>video_id</u>	timeuuid
added_date	timestamp
description	text
title	text
user_id	uuid

# Esercizio 1 - soluzione

---

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

```
USE killrvideo;
```

```
CREATE TABLE videos(
  video_id TIMEUUID,
  added_date TIMESTAMP,
  description TEXT,
  title TEXT,
  user_id uuid,
  PRIMARY KEY (video_id)
);
```

```
COPY videos
```

```
FROM '/root/labwork/exercise-2/videos.csv'
WITH HEADER=true;
```

```
SELECT *
FROM videos;
```

```
SELECT COUNT(*)
FROM videos;
```

```
SELECT *
FROM videos
LIMIT 10;
```

# Limitazioni in fase di query

---

Cassandra impedisce di filtrare una tabella senza specificare l'ID

- A meno che non sia stato costruito un indice secondario

```
CREATE TABLE videos(  
  video_id TIMEUUID,  
  added_date TIMESTAMP,  
  description TEXT,  
  title TEXT,  
  user_id uuid,  
  PRIMARY KEY (video_id)  
);
```

```
SELECT *  
FROM videos  
WHERE title = 'The Original Grumpy Cat';
```

```
SELECT *  
FROM videos  
WHERE added_date < '2015-05-01';
```

```
cqlsh:killrvideo> select * from videos where title = 'The Original Grumpy Cat';  
InvalidRequest: code=2200 [Invalid query] message="No secondary indexes on the r  
estricted columns support the provided operators: "  
cqlsh:killrvideo> select * from videos where added_date < '2015-05-01';  
InvalidRequest: code=2200 [Invalid query] message="No secondary indexes on the r  
estricted columns support the provided operators: "
```

# Esercizio 2

---

- Creare una nuova tabella per memorizzare video con una chiave composta
  - videos\_by\_title\_year
  - E' buona pratica dare alle tabelle un nome che rispecchi il modo di interrogarle
  - /root/labwork/exercise-3/videos\_by\_title\_year.csv
- Testare qualche query
  - Cercare un video per nome e anno
  - Cercare video solo per nome o solo per anno

Column Name	Data Type
<u>title</u>	text
<u>added_year</u>	int
added_date	timestamp
description	text
user_id	uuid
video_id	uuid



# Esercizio 2 - soluzione

---

```
CREATE TABLE videos_by_title_year(  
  title TEXT,  
  added_year int,  
  added_date TIMESTAMP,  
  description TEXT,  
  user_id uuid,  
  video_id TIMEUUID,  
  PRIMARY KEY ((title, added_year))  
);
```

```
COPY videos_by_title_year  
FROM '/root/labwork/exercise-3/videos_by_title_year.csv'  
WITH HEADER=true;
```

```
SELECT *  
FROM videos_by_title_year  
WHERE title = 'Sleepy Grumpy Cat'  
AND added_year = 2015;
```

```
SELECT *  
FROM videos_by_title_year  
WHERE added_year = 2015;
```

# Chiavi

---

CHIAVE PRIMARIA, CHIAVE DI PARTIZIONAMENTO

# Chiave primaria

---

## Chiave primaria

- Insieme di campi i cui valori identificano la riga all'interno della tabella
- Stessa semantica del modello relazionale

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

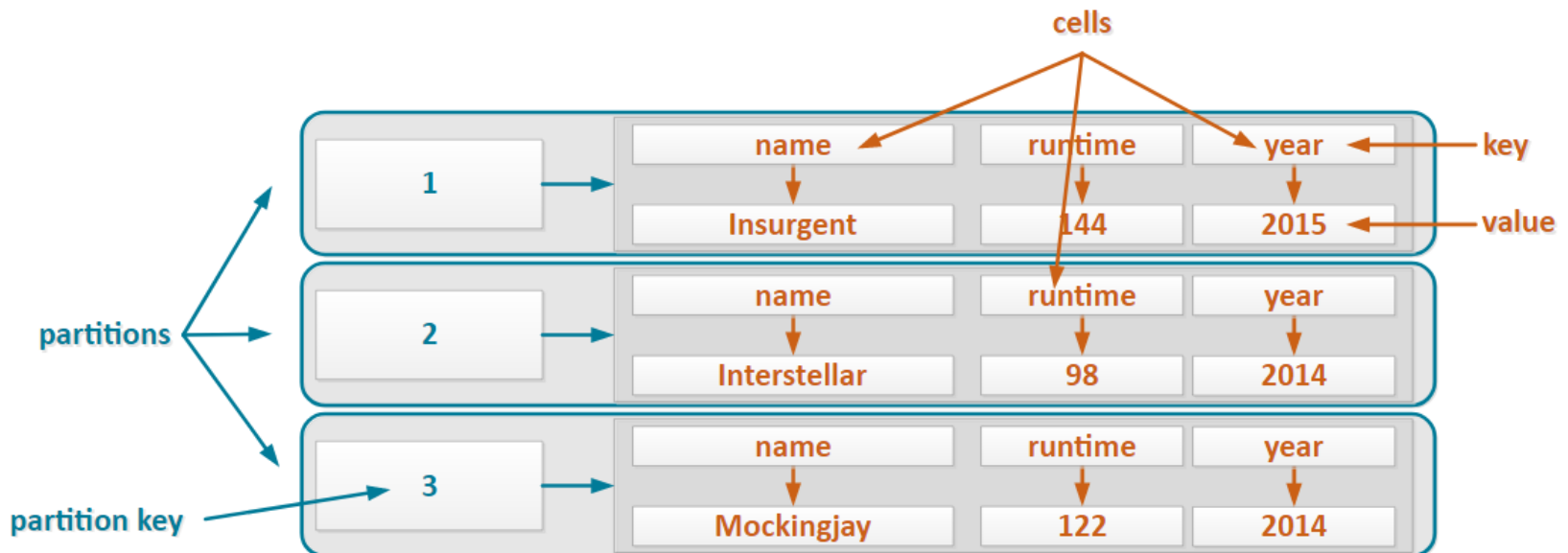
id	name	runtime	year
1	Insurgent	119	2015
2	Interstellar	98	2014
3	Mockingjay	122	2014

# Cassandra al livello fisico

Le righe sono memorizzate in blocchi detti **partizioni**

Le partizioni sono definite sulla base di uno o più campi, definiti **chiave di partizione**

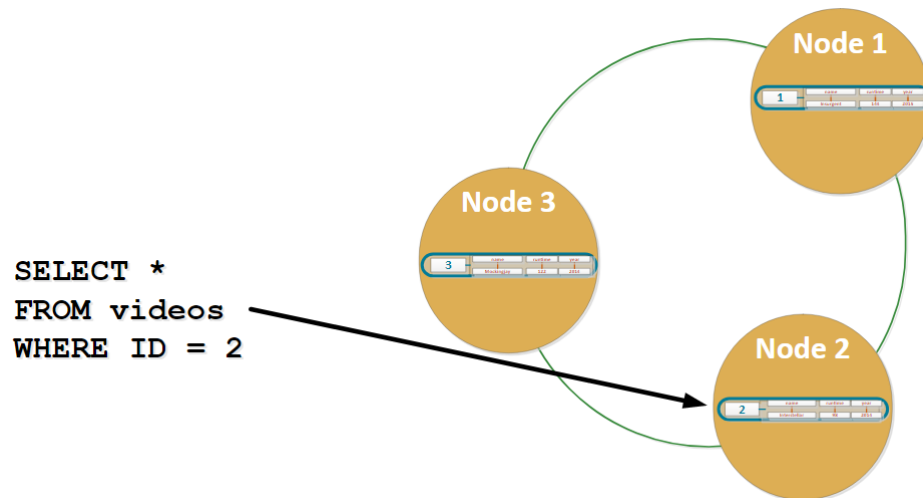
- Di default, la chiave primaria funge da chiave di partizione



# Perchè il concetto di partizione

La partizione è il blocco di dati che viene distribuito

- Si usa una funzione di hashing sulla chiave di partizione per decidere in quale nodo salvare la partizione...
- ... e anche per sapere quale nodo interrogare per ritrovarla se so la chiave (senza doverli interrogare tutti)



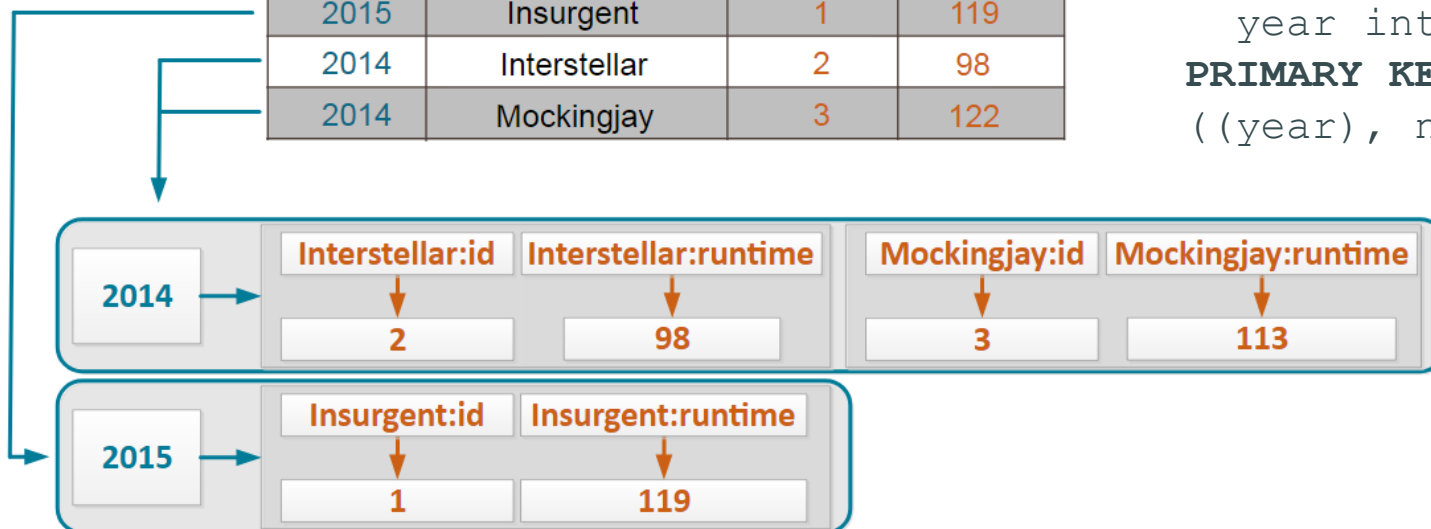
# Chiave di partizione

La chiave di partizione corrisponde alla chiave primaria o ad un suo **sottoinsieme**

- Se la chiave di partizionamento è un sottoinsieme della chiave primaria, **più righe possono essere raggruppate nella stessa partizione**

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

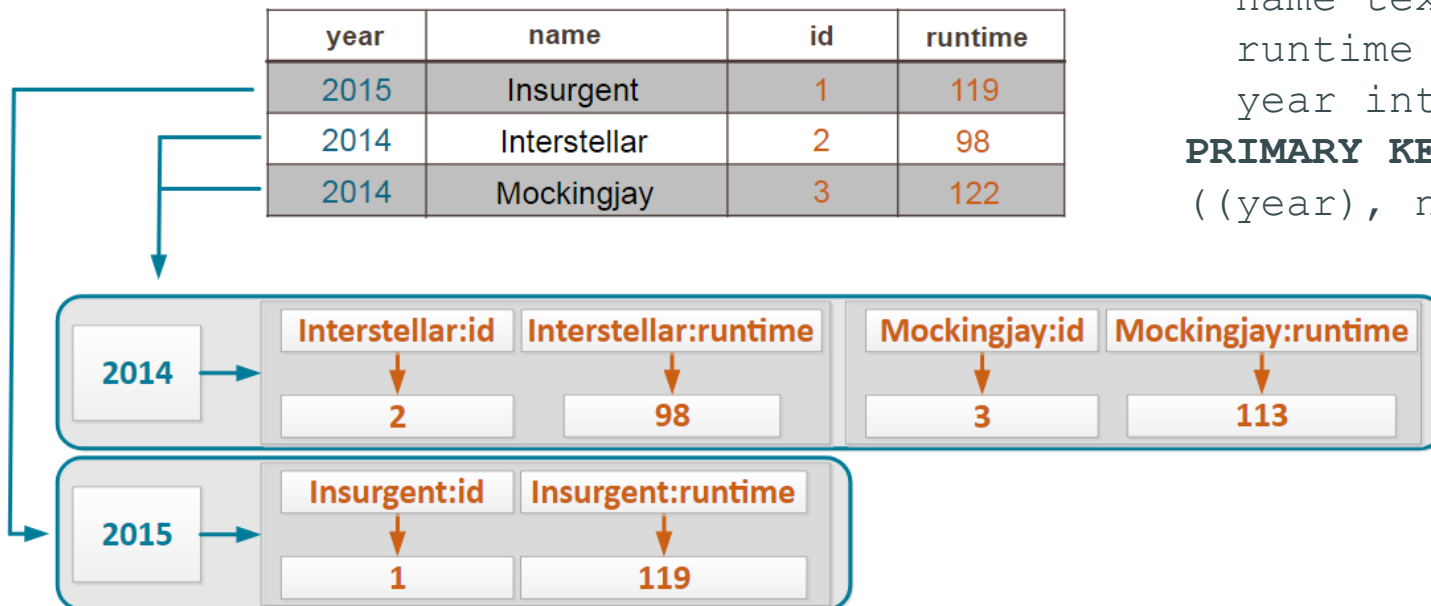
year	name	id	runtime
2015	Insurgent	1	119
2014	Interstellar	2	98
2014	Mockingjay	3	122



# Chiave di partizione

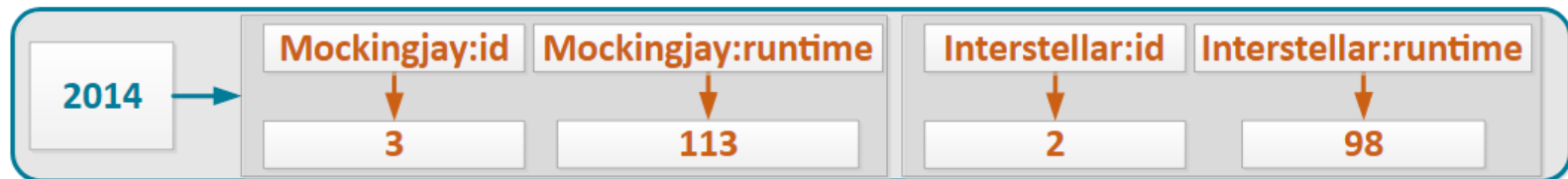
I campi della chiave primaria che non fanno parte della chiave di partizionamento vengono definite **colonne di raggruppamento** e **definiscono l'ordine** con cui le righe sono memorizzate all'interno della partizione

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



# Colonna di raggruppamento

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

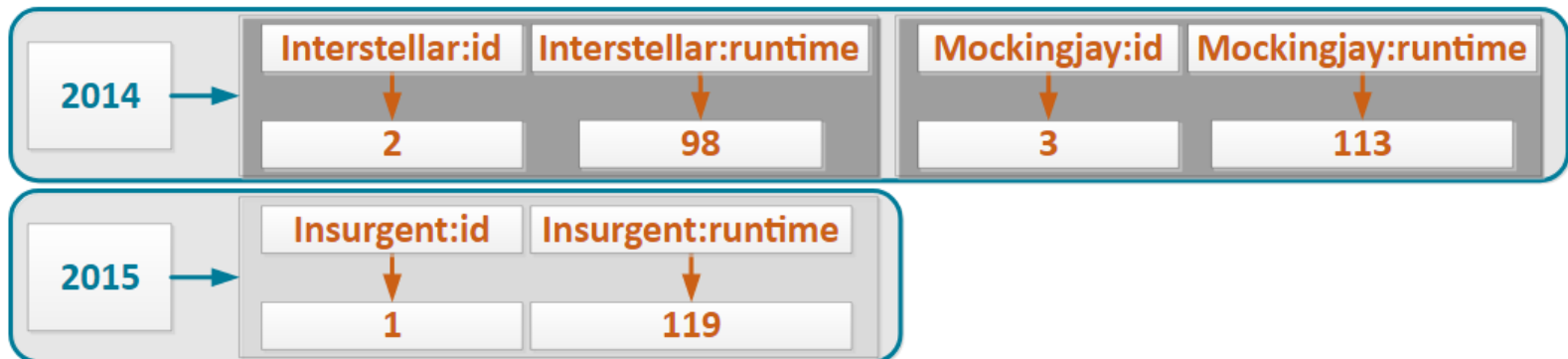




# Colonna di raggruppamento

L'ordinamento delle righe consente di effettuare query di range

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



# Morale

---

E' possibile sfruttare i concetti di chiave primaria e chiave di partizione per definire le tabelle in funzione...

- ... non solo dei dati che deve contenere...
- ... ma anche di come li voglio interrogare

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

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

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

# Esercizio 3

---

- Creare una nuova tabella per memorizzare video con una chiave composta diversa
  - Obiettivo: [interrogare la tabella sulla base dei campi tag e year e fare range queries su year](#)
  - videos\_by\_tag\_year
  - /root/labwork/exercise-4/videos\_by\_tag\_year.csv
  - ATTENZIONE: un video può essere associate a tanti tag

Column Name	Data Type
<u>tag</u>	text
added_year	int
<u>video_id</u>	timeuuid
added_date	timestamp
description	text
title	text
user_id	uuid

# Esercizio 3

---

```
CREATE TABLE videos_by_tag_year(  
  tag TEXT,  
  added_year int,  
  added_date TIMESTAMP,  
  title TEXT,  
  description TEXT,  
  user_id uuid,  
  video_id TIMEUUID,  
  PRIMARY KEY ((video_id, tag))  
);
```

Posso avere più tag per lo stesso video? SI

Posso interrogare per tag e year? NO

Posso fare query di range su added\_year? NO

# Esercizio 3

---

```
CREATE TABLE videos_by_tag_year(  
  tag TEXT,  
  added_year int,  
  added_date TIMESTAMP,  
  title TEXT,  
  description TEXT,  
  user_id uuid,  
  video_id TIMEUUID,  
  PRIMARY KEY ((tag), added_year)  
) WITH CLUSTERING ORDER BY (added_year DESC);
```

Posso avere più tag per lo stesso video? SI

Posso interrogare per tag e year? SI

Posso fare query di range su added\_year? SI

Posso avere più video nello stesso anno con lo stesso tag? NO

# Esercizio 3

---

```
CREATE TABLE videos_by_tag_year(  
  tag TEXT,  
  added_year int,  
  added_date TIMESTAMP,  
  title TEXT,  
  description TEXT,  
  user_id uuid,  
  video_id TIMEUUID,  
  PRIMARY KEY ((tag), added_year, video_id)  
) WITH CLUSTERING ORDER BY (added_year DESC);
```

Posso avere più tag per lo stesso video? SI

Posso interrogare per tag e year? SI

Posso fare query di range su added\_year? SI

Posso avere più video nello stesso anno con lo stesso tag? SI

# Esercizio 3

---

```
COPY videos_by_tag_year
FROM '/root/labwork/exercise-4/videos_by_tag_year.csv'
WITH HEADER=true;
```

```
SELECT * FROM videos_by_tag_year
WHERE tag= 'trailer' AND added_year = 2015;
```

```
SELECT * FROM videos_by_tag_year
WHERE tag= 'cql' AND added_year < 2015;
```

```
SELECT * FROM videos_by_tag_year
WHERE added_year < 2015;
```

```
SELECT * FROM videos_by_tag_year
WHERE tag= 'cql';
```

# Tipi di dato complessi

---



# DDL

---

Rimuovere tutte le righe in una tabella

```
TRUNCATE table1;
```

Aggiungere una colonna

```
ALTER TABLE table1 ADD another_column text;
```

Rimuovere una colonna

```
ALTER TABLE table1 DROP another_column;
```

Non si può modificare la chiave primaria

# Collezioni

---

Le collezioni sono colonne che contengono più valori

Progettate per memorizzare un insieme limitato di dati

Se interrogate, restituiscono il contenuto intero

Non innestabili (non esistono collezioni di collezioni)

# Tipi di collezione

---

## Set

- Valori univoci e ordinati

**SET**<TEXT>

## List

- Valori non-univoci e ordinati

LIST<TEXT>

## Map

- Coppie chiave-valore ordinate per chiave (univoca)

MAP<TEXT, INT>

## UDT (user-defined type)

- Permette di creare colonne più complesse

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

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

# Come usare UDT

---

```
CREATE TABLE users (  
  id uuid,  
  first_name frozen <full_name>,  
  direct_reports set<frozen <full_name>>,  
  addresses map<text, frozen <address>>,  
  PRIMARY KEY ((id))  
);
```

La clausola frozen consente di considerare il campo come un blocco atomico

- Per aggiornare una porzione del campo (e.g., solo first\_name) bisogna aggiornare tutto il campo
- Il campo è trattato come un BLOB
- E' possibile non specificare la clausola e lasciare il campo non-frozen
- Il campo DEVE essere frozen se lo si vuole utilizzare nella chiave primaria

# Contatori

---

I contatori sono un tipo di dato speciale offerto da Cassandra

- Permettono di evitare problemi di concorrenza nell'update di un campo
- Inizializzati a 0
- Si possono aumentare o decrementare, ma non si possono fare assegnamenti diretti su di essi
- Se usati, non possono esistere colonne non-chiave che non siano contatori

```
CREATE TABLE moo_counts (  
    cow_name text,  
    moo_count counter,  
    PRIMARY KEY ((cow_name))  
);
```

```
UPDATE moo_counts  
SET moo_count = moo_count + 8  
WHERE cow_name = 'Betsy';
```

# Esercizio 4

---

Obiettivo: estendere la tabella originale *videos* con:

- Una colonna *tags* come collezione di stringhe
  - Troncare la tabella e ripopolarla caricando i dati che comprendono i tag
  - Fonte: `/root/labwork/exercise-5/videos.csv`
- Una colonna *video\_encoding* di tipo UDT e così composta
  - Utilizzare sempre il comando `copy` per popolare solamente i valori della nuova colonna
  - Fonte: `/root/labwork/exercise-5/videos_encoding.csv`

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

# Esercizio 4

---

```
TRUNCATE videos;
```

```
ALTER TABLE videos ADD tags SET<TEXT>;
```

```
COPY videos FROM '/root/labwork/exercise-5/videos.csv' WITH HEADER=true;
```

```
CREATE TYPE video_encoding (  
    bit_rates SET<TEXT>,  
    encoding text,  
    height int,  
    width int  
);
```

```
ALTER TABLE videos ADD encoding FROZEN<video_encoding>;
```

```
COPY videos (video_id, encoding)  
FROM '/root/labwork/exercise-5/videos_encoding.csv'  
WITH HEADER=true;
```

```
SELECT * FROM videos LIMIT 10;
```

# Eseguire script

---

Uno script può contenere tante istruzioni CQL

```
SOURCE './myscript.cql';
```



# Esercizio 5

---

Obiettivo: creare una tabella *videos\_count\_by\_tag* con un contatore che conteggi il numero di video a cui è assegnato un determinato tag

- Fonte: `/root/labwork/exercise-6/videos_count_by_tag.cql`
  - Notare che il file è `.cql`, non `.csv`
  - Notare che il file contiene solo `update`, nessuna `insert`
- Prima di creare la tabella, verificare le colonne presenti nel file `.cql`

# Esercizio 5

---

```
CREATE TABLE videos_count_by_tag(  
    tag TEXT,  
    added_year int,  
    video_count COUNTER,  
    PRIMARY KEY ((tag), added_year)  
);
```

```
SOURCE '/root/labwork/exercise-6/videos_count_by_tag.cql';
```

```
SELECT * FROM videos_count_by_tag LIMIT 5;
```

```
UPDATE videos_count_by_tag  
SET video_count = video_count + 10  
WHERE tag = 'You Are Awesome' AND added_year = 2015;
```

```
SELECT * FROM videos_count_by_tag  
WHERE tag = 'You Are Awesome';
```

# Modellazione

---

# Modellazione relazionale

---

**videos**

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

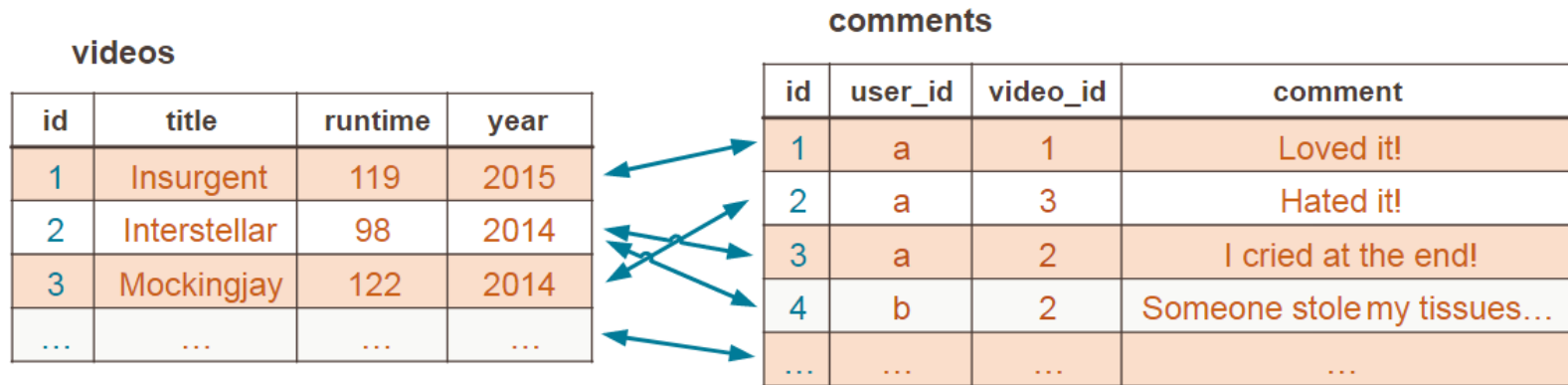
**users**

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...
...	...	...	...

# Query: commenti di un video



```
SELECT comment
FROM videos JOIN comments
  ON videos.id = comments.video_id
WHERE title = 'Interstellar'
```

# Query: commenti di un video

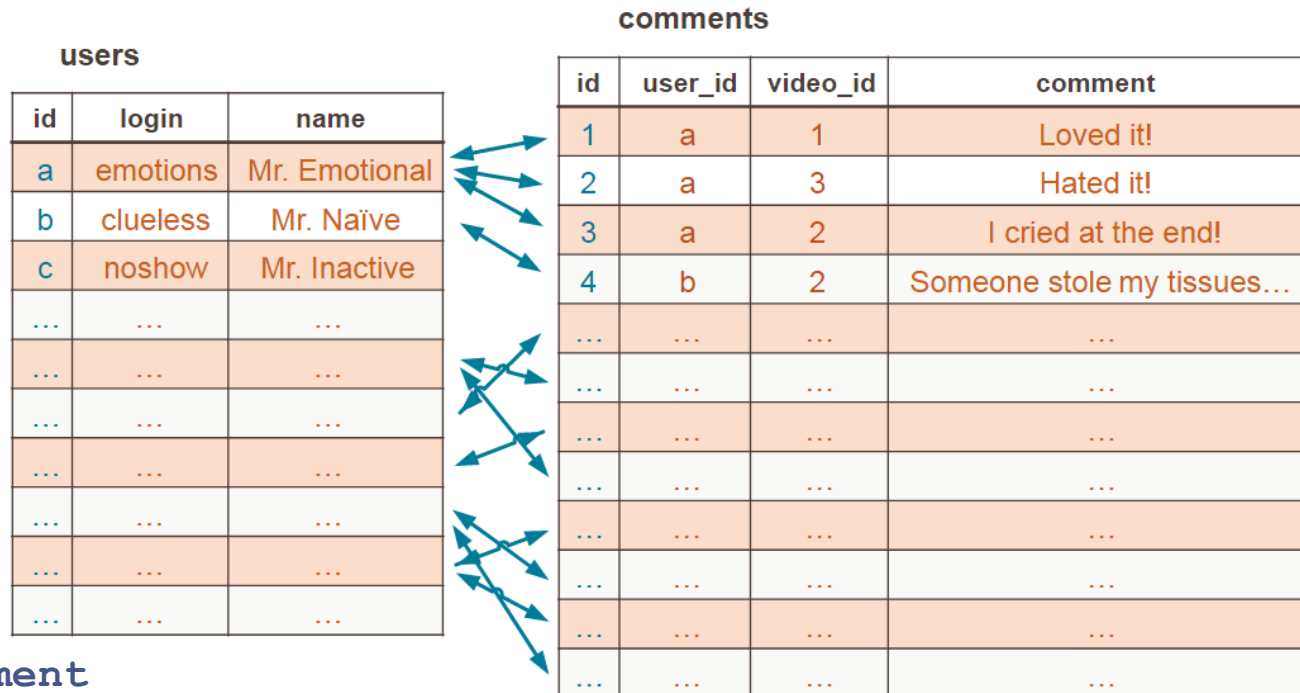
videos JOIN comments

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...
...	...	...	...	...	...	...	...

WHERE title = 'Interstellar'

id	title	runtime	year	id	user_id	video_id	comment
2	Interstellar	98	2014	3	a	2	I cried at the end!
2	Interstellar	98	2014	4	b	2	Someone stole my tissues...

# Query: commenti di un utente



```
SELECT comment
FROM users JOIN comments
  ON users.id = comments.user_id
WHERE user.login = 'emotions'
```

# Modellazione in Cassandra

---

In Cassandra non si possono fare join

- Soluzione: denormalizzare!

Come denormalizzare?

- Dipende fondamentalmente dalle query che si vogliono fare

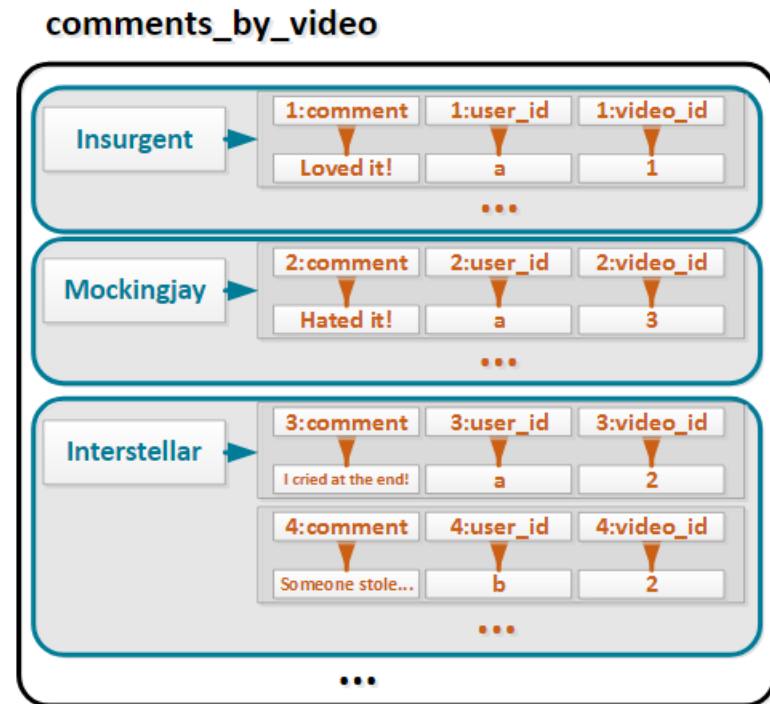
Se si vogliono fare tante query diverse?

- Q1: cerca i commenti fatti su un video
- Q2: cerca i commenti fatti da un utente
- Soluzione: duplicare i commenti in tabelle diverse



# Query: commenti di un video

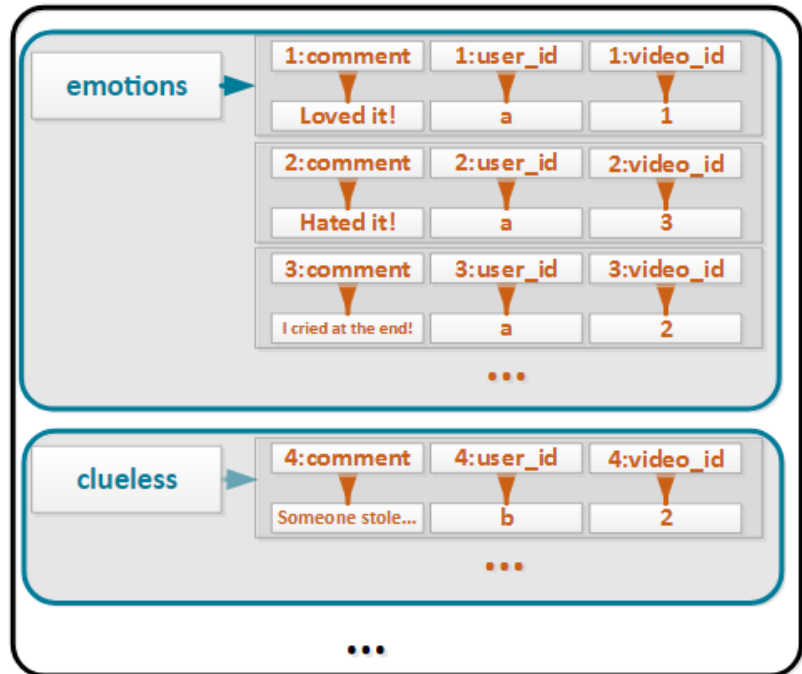
```
CREATE TABLE comments_by_video (  
  video_title text,  
  comment_id timeuuid,  
  user_id text,  
  video_id timeuuid,  
  comment text,  
  PRIMARY KEY ((video_title), comment_id)  
);
```



# Query: commenti di un utente

```
CREATE TABLE comments_by_user (  
  user_login text,  
  comment_id timeuuid,  
  user_id text,  
  video_id timeuuid,  
  comment text,  
  PRIMARY KEY ((user_login), comment_id) );
```

comments\_by\_user



# Esercizio 6

---

Obiettivo: modellare la relazione tra video e attori

- La modellazione deve supportare le query Q1 e Q2

## Video

Column Name	Data Type
video_id	timeuuid
added_date	timestamp
description	text
encoding	video_encoding
tags	set<text>
title	text
user_id	uuid

## Attori

Column Name	Data Type
actor	text
character	text
genre	text

Q1: restituire i video in cui compare un dato attore (a partire dal più recente)

Q2: restituire i video di un dato genere (a partire dal più recente)

# Esercizio 6

---

Q1: restituire i video in cui compare un dato attore (a partire dal più recente)

- Filtro su attore → l'attore costituisce la chiave di partizionamento
- Ordinamento per data → la data costituisce la chiave di raggruppamento
- Restituire i video → bisogna mantenere la granularità dei video

```
CREATE TABLE videos_by_actor (  
  actor text,  
  added_date timestamp,  
  video_id timeuuid,  
  character_name text,  
  description text,  
  encoding frozen<video_encoding>,  
  tags set<text>,  
  title text,  
  user_id uuid,  
  PRIMARY KEY ( ?? )  
) WITH CLUSTERING ORDER BY ( ?? );
```

# Esercizio 6

---

Q1: restituire i video in cui compare un dato attore (a partire dal più recente)

```
CREATE TABLE videos_by_actor (  
  actor text,  
  added_date timestamp,  
  video_id timeuuid,  
  character_name text,  
  description text,  
  encoding frozen<video_encoding>,  
  tags set<text>,  
  title text,  
  user_id uuid,  
  PRIMARY KEY ((actor), added_date,  
    video_id, character_name)  
) WITH CLUSTERING ORDER BY (  
  added_date DESC,  
  video_id ASC,  
  character_name ASC  
) ;
```

```
COPY videos_by_actor  
FROM '/root/labwork/exercise-7/videos_by_actor.csv'  
WITH HEADER=true;
```

```
SELECT *  
FROM videos_by_actor  
WHERE actor = 'Tom Hanks';
```

```
SELECT actor, added_date  
FROM videos_by_actor  
WHERE actor = 'Tom Hanks';
```

# Esercizio 6

---

Q2: restituire i video di un dato genere (a partire dal più recente)

- Filtro su genere → il genere costituisce la chiave di partizionamento
- Ordinamento per data → la data costituisce la chiave di raggruppamento
- Restituire i video → bisogna mantenere la granularità dei video

```
CREATE TABLE videos_by_genre (  
    genre text,  
    added_date timestamp,  
    video_id timeuuid,  
    description text,  
    encoding frozen<video_encoding>,  
    tags set<text>,  
    title text,  
    user_id uuid,  
    PRIMARY KEY ( ?? )  
) WITH CLUSTERING ORDER BY ( ?? );
```

# Esercizio 6

---

Q2: restituire i video di un dato genere (a partire dal più recente)

```
CREATE TABLE videos_by_genre (  
  genre text,  
  added_date timestamp,  
  video_id timeuuid,  
  description text,  
  encoding frozen<video_encoding>,  
  tags set<text>,  
  title text,  
  user_id uuid,  
  PRIMARY KEY ((genre), added_date,  
    video_id)  
) WITH CLUSTERING ORDER BY (  
  added_date DESC,  
  video_id ASC  
);
```

```
COPY videos_by_genre  
FROM '/root/labwork/exercise-7/videos_by_genre.csv'  
WITH HEADER=true;
```

```
SELECT *  
FROM videos_by_genre  
WHERE genre = 'Time travel';
```

# Upsert

---

Nei DBMS relazionali (ma anche in MongoDB) la funzionalità di UPSERT (i.e., inserisci se la tupla/il documento non esiste già, altrimenti aggiornane i valori) è gestita con comandi ad-hoc

- `db.collection.update( queryObj, updateObj, { upsert: true })`
- Inserire un documento con un ID già esistente dà errore
- Aggiornare un documento con un ID non esistente non ha alcun effetto

In Cassandra l'UPSERT è il comportamento di default

- Inserire una riga con un ID già esistente significa aggiornarne i valori
- Aggiornare una riga con un ID non esistente significa aggiungerla



# Esercizio 3

---

```
COPY bad_videos_by_tag_year (tag, added_year, video_id, added_date,  
description, title, user_id)  
FROM '/root/labwork/exercise-4/videos_by_tag_year.csv'  
WITH HEADER=true;
```

Number of inserted rows: 797

```
SELECT count(*)  
FROM bad_videos_by_tag_year ;
```

Un video può avere più tag, quindi i suoi dati sono duplicati nel CSV (tante righe, una per ogni tag)  
Ma il comportamento di default in Cassandra è l'upsert..

```
DROP TABLE bad_videos_by_tag_year ;
```