Cassandra

Principais equívocos na modelagem de dados

Henrique Menezes

TEMPEST
Professional Services
THE DEVELOPER'S CONFERENCE

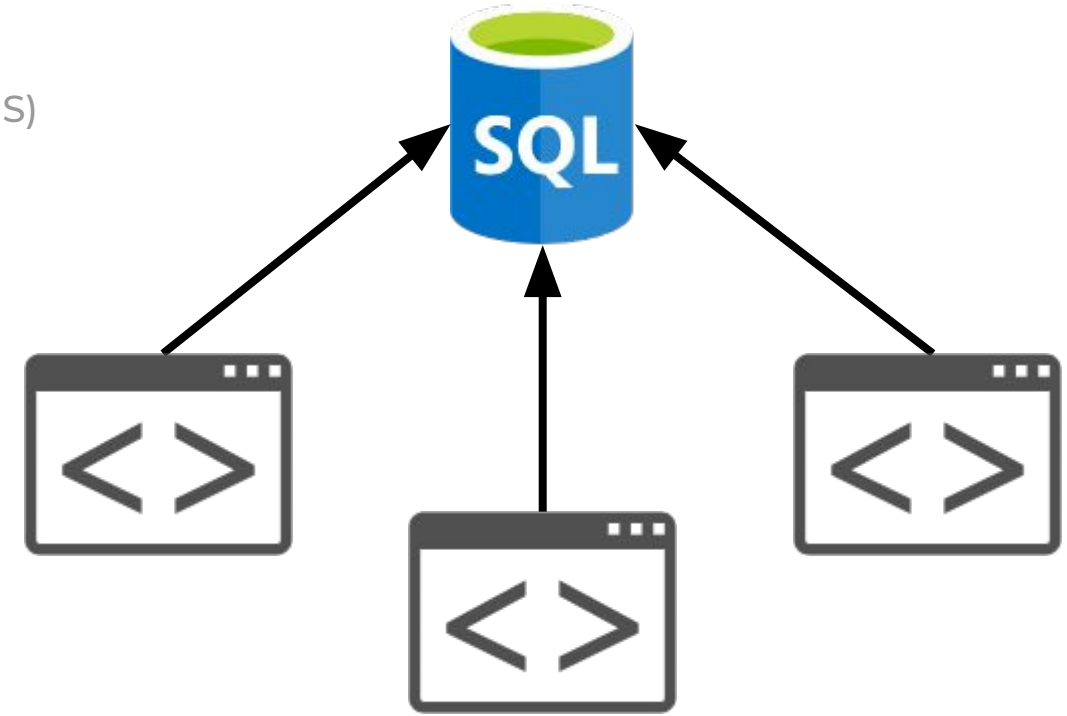
Aplicações tradicionais

Banco de Dados Relacionais (RDBMS)

Crescimento da aplicação

Novas necessidades:

- Escalabilidade
- Disponibilidade



RDBMS NÃO ESCALA!

RDBMS escala, porém..

RDBMS - Vertical Scaling

Máquinas maiores

Custo mais elevado

Único Ponto de Falha!

Manutenção e atualização

2 vCPU
8 GB
100 GB
$140.00

8 vCPU
32 GB
500 GB
$580.00

16 vCPU
64 GB
2000 GB
$1400.00

RDBMS - Horizontal Scaling (Master-Slave)

Master-Slave

Réplicas de Leitura

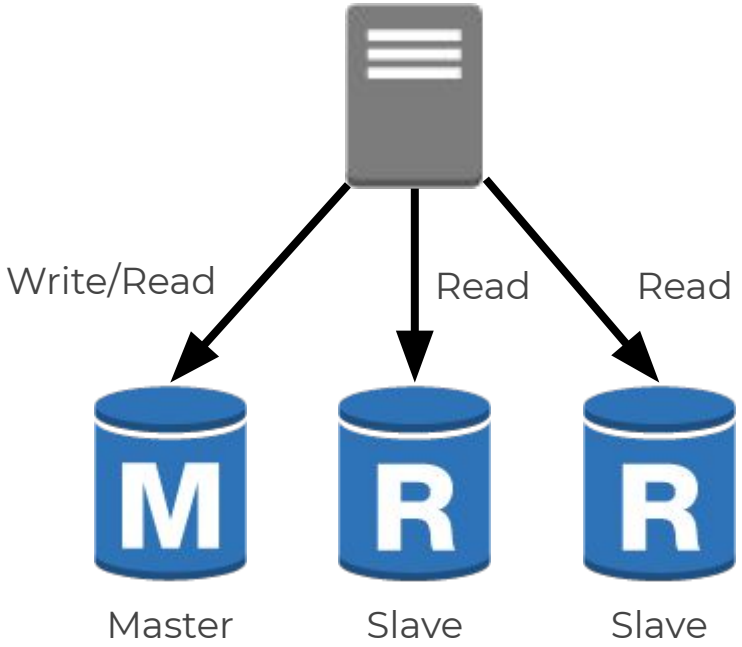
Distribuir a Carga de Leitura

Replicação Assíncrona

~~Consistência~~

~~Isolamento~~

~~Atomicidade~~



RDBMS - Horizontal Scaling (Sharding)

Sharding

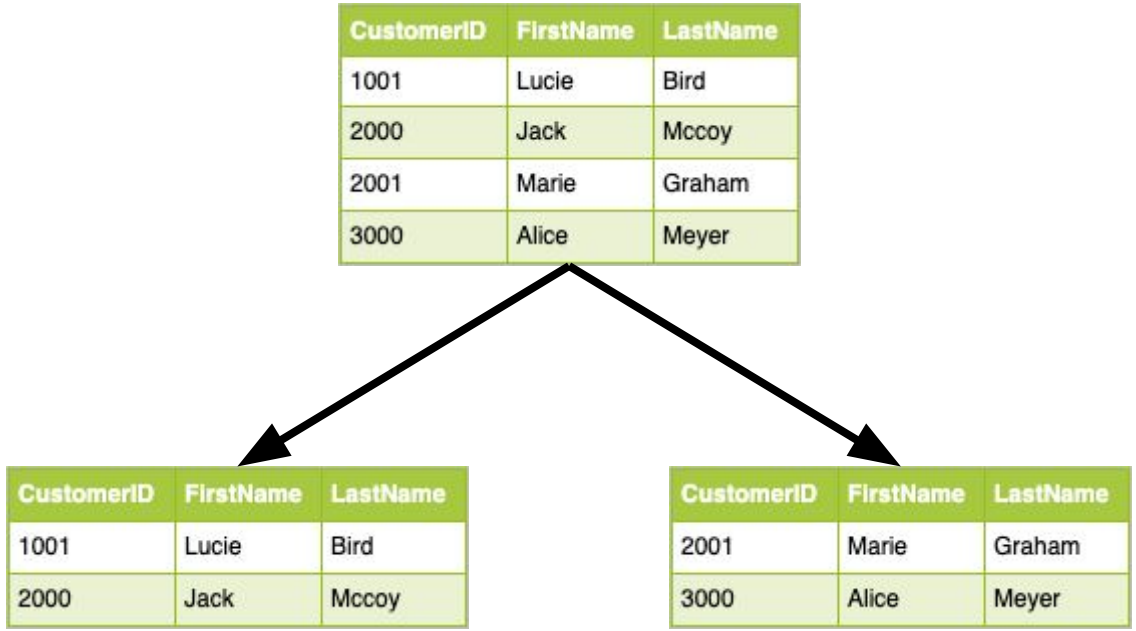
Particiona os dados

~~JOIN~~

~~Normalização (3FN)~~

Locking Excessivo

~~Disponibilidade~~



CustomerID	FirstName	LastName
1001	Lucie	Bird
2000	Jack	Mccoy
2001	Marie	Graham
3000	Alice	Meyer

CustomerID	FirstName	LastName
1001	Lucie	Bird
2000	Jack	Mccoy

CustomerID	FirstName	LastName
2001	Marie	Graham
3000	Alice	Meyer

ACID e 3FN NÃO ESCALAM!

Em busca de uma solução..

NoSQL databases

Agora é só migrar o modelo **RDBMS** para o banco **NoSQL...**

Não é bem assim.

1. Cada BANCO tem seu PROPÓSITO

Document

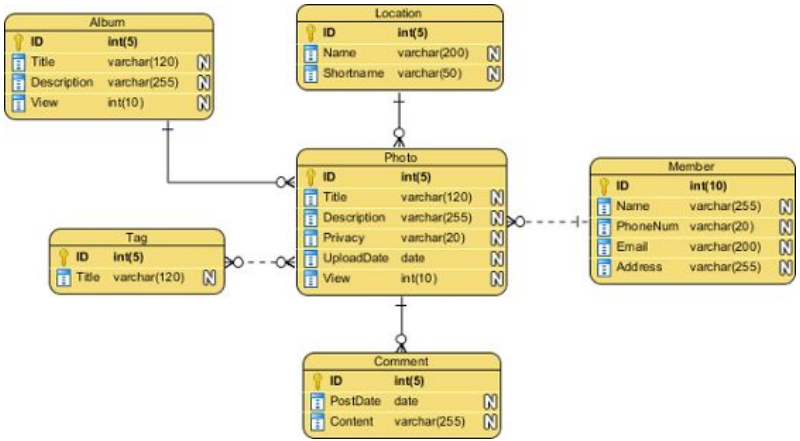
Key-Value

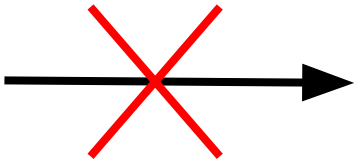
Wide Column

Graph

Search Engine

2. O MODELO não é compatível





RDBMS **NoSQL**

O que é Cassandra?

Origem

Criado pelo Facebook, e por um dos co-autores do Dynamo.

Baseado no DynamoDB (AWS) e BigTable (Google).

2008 - Open Source

2009 - Apache Incubator

2010 - Top-Level Project no ASF

O que é Cassandra?

NoSQL

Distribuído

Altamente Escalável

Grande quantidade de dados

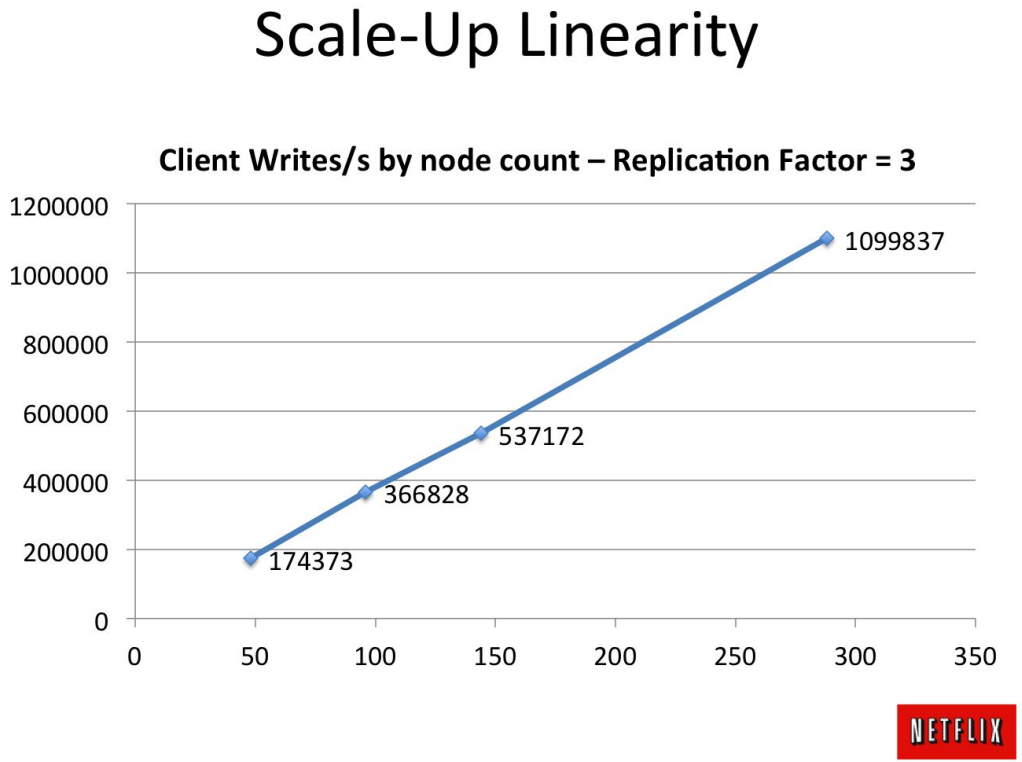
Commodity Hardware

Alta Disponibilidade

Sem SPOF (Single Point of Failure)

Escalabilidade Linear

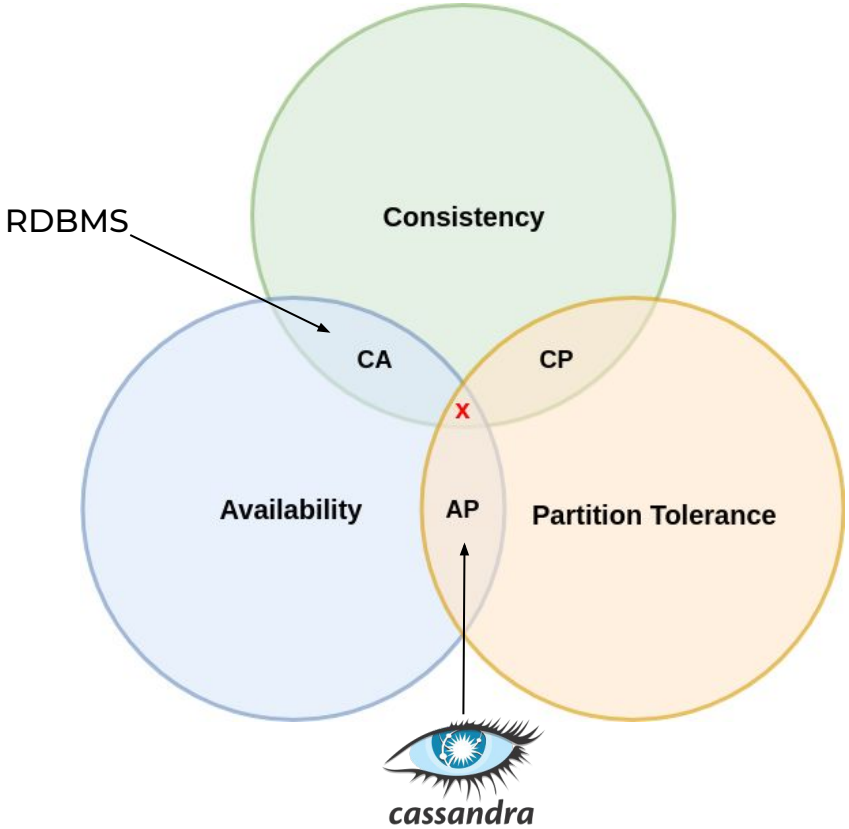
Escalabilidade Linear



NETFLIX

TEMPEST
Protegendo negócios no mundo digital.

CAP Theorem


RDBMS
Consistency
CA
CP
X
Availability
AP
Partition Tolerance
cassandra

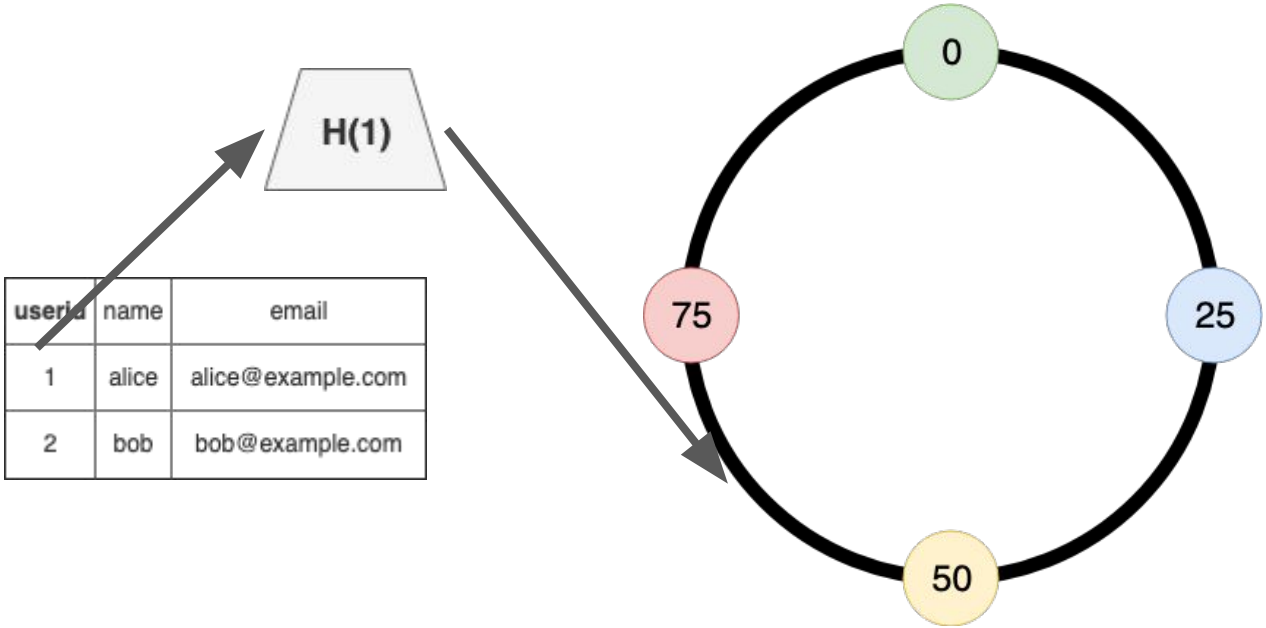
Conceitos Básicos

Hash Ring

Partition key

Clustering column

Row key



Principais equívocos

Primary Key

Primary Key

Identificar unicamente uma linha

```
CREATE TABLE users (
    email text,
    firstname text,
    lastname text,
    password text,
    PRIMARY KEY (email)
);
```


```
demo=# INSERT INTO users (email, firstname, lastname, password)
demo-# VALUES ('john.doe@example.com', 'John', 'Doe', '123456');
INSERT 0 1
demo=#
demo=#
demo=# INSERT INTO users (email, firstname, lastname, password)
demo-# VALUES ('john.doe@example.com', 'John', 'Doe', 'password');
ERROR:  duplicate key value violates unique constraint "users_pkey"
DETAIL:  Key (email)=(john.doe@example.com) already exists.
demo=#
demo=#
demo=# select * from users;
        email         | firstname | lastname | password
----------------------+-----------+----------+----------
 john.doe@example.com | John      | Doe      | 123456
(1 row)
```

RDBMS: Valida a constraint unique

Primary Key

Identificar unicamente uma linha

```
CREATE TABLE users (
    email text,
    firstname text,
    lastname text,
    password text,
    PRIMARY KEY (email)
);
```


```
cqlsh:demo> INSERT INTO users (email, firstname, lastname, password)
        ... VALUES ('john.doe@example.com', 'John', 'Doe', '123456');
cqlsh:demo>
cqlsh:demo> INSERT INTO users (email, firstname, lastname, password)
        ... VALUES ('john.doe@example.com', 'John', 'Doe', 'password');
cqlsh:demo>
cqlsh:demo> select * from users;

 email                | firstname | lastname | password
----------------------+-----------+----------+----------
 john.doe@example.com |      John |      Doe | password

(1 rows)
```

Cassandra: Não valida a constraint unique

Primary Key

Não valida a constraint unique

Prioriza performance

INSERT e UPDATE = UPSERT

```
cqlsh:demo> UPDATE users
        ... SET firstname = 'Alice', lastname = 'Apple', password = 'mypassword'
        ... WHERE email = 'alice.apple@example.com';
cqlsh:demo>
cqlsh:demo> select * from users;

 email                   | firstname | lastname | password
-------------------------+-----------+----------+------------
    john.doe@example.com |      John |      Doe |   password
 alice.apple@example.com |     Alice |    Apple | mypassword

(2 rows)
```

Tabela única por entidade

Tabela única por entidade

Tabela única com o identificador (PK)

Índice nas colunas para permitir consultas


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

CREATE INDEX videos_title_idx ON videos(title);
CREATE INDEX videos_year_idx ON videos(year);
```

```
demo=# SELECT * FROM videos;
 id |       title        | runtime | year
----+--------------------+---------+------
  1 | TDC: MongoDB       |      30 | 2018
  2 | TDC: ElasticSearch |      40 | 2018
  3 | TDC: Cassandra     |      50 | 2019
(3 rows)

demo=# SELECT * FROM videos WHERE title = 'TDC: Cassandra';
 id |     title      | runtime | year
----+----------------+---------+------
  3 | TDC: Cassandra |      50 | 2019
(1 row)

demo=# SELECT * FROM videos WHERE year = 2018;
 id |       title        | runtime | year
----+--------------------+---------+------
  1 | TDC: MongoDB       |      30 | 2018
  2 | TDC: ElasticSearch |      40 | 2018
(2 rows)
```

Tabela única por entidade

Tabela única com o identificador (PK)

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

```
cqlsh:demo> SELECT * FROM videos;

 id | runtime | title              | year
----+---------+--------------------+------
  1 |      30 |       TDC: MongoDB | 2018
  2 |      40 | TDC: ElasticSearch | 2018
  3 |      50 |     TDC: Cassandra | 2019

(3 rows)
cqlsh:demo> SELECT * FROM videos WHERE title = 'TDC: Cassandra';
InvalidRequest: Error from server: code=2200 [Invalid query] message
="Cannot execute this query as it might involve data filtering and t
hus may have unpredictable performance. If you want to execute this
query despite the performance unpredictability, use ALLOW FILTERING"
cqlsh:demo> SELECT * FROM videos WHERE year = 2018;
InvalidRequest: Error from server: code=2200 [Invalid query] message
="Cannot execute this query as it might involve data filtering and t
hus may have unpredictable performance. If you want to execute this
query despite the performance unpredictability, use ALLOW FILTERING"
```

Tabela única por entidade

ALLOW FILTERING


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

```
cqlsh:demo> SELECT * FROM videos WHERE title = 'TDC: Cassandra' ALLOW FILTERING;

 id | runtime | title          | year
----+---------+----------------+------
  3 |      50 | TDC: Cassandra | 2019

(1 rows)
cqlsh:demo> SELECT * FROM videos WHERE year = 2018 ALLOW FILTERING;

 id | runtime | title              | year
----+---------+--------------------+------
  1 |      30 |       TDC: MongoDB | 2018
  2 |      40 | TDC: ElasticSearch | 2018

(2 rows)
```

Não use ALLOW FILTERING!!

Tabela única por entidade

Secondary Index

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

CREATE INDEX videos_title_idx ON videos(title);
CREATE INDEX videos_year_idx ON videos(year);
```

```
cqlsh:demo> CREATE INDEX videos_title_idx ON videos(title);
cqlsh:demo> CREATE INDEX videos_year_idx ON videos(year);
cqlsh:demo>
cqlsh:demo> SELECT * FROM videos WHERE title = 'TDC: Cassandra';

 id | runtime | title          | year
----+---------+----------------+------
  3 |      50 | TDC: Cassandra | 2019

(1 rows)
cqlsh:demo> SELECT * FROM videos WHERE year = 2018;

 id | runtime | title              | year
----+---------+--------------------+------
  1 |      30 |       TDC: MongoDB | 2018
  2 |      40 | TDC: ElasticSearch | 2018

(2 rows)
```

Tabela única por entidade

Secondary Index

Problemas:

Consulta o índice em todos os nós

Colunas com alta cardinalidade

Colunas frequentemente atualizadas

Consulta em partição grande

```
cqlsh:demo> CREATE INDEX videos_title_idx ON videos(title);
cqlsh:demo> CREATE INDEX videos_year_idx ON videos(year);
cqlsh:demo>
cqlsh:demo> SELECT * FROM videos WHERE title = 'TDC: Cassandra';

 id | runtime | title          | year
----+---------+----------------+------
  3 |      50 | TDC: Cassandra | 2019

(1 rows)
cqlsh:demo> SELECT * FROM videos WHERE year = 2018;

 id | runtime | title              | year
----+---------+--------------------+------
  1 |      30 |       TDC: MongoDB | 2018
  2 |      40 | TDC: ElasticSearch | 2018

(2 rows)
```

Tabela única por entidade

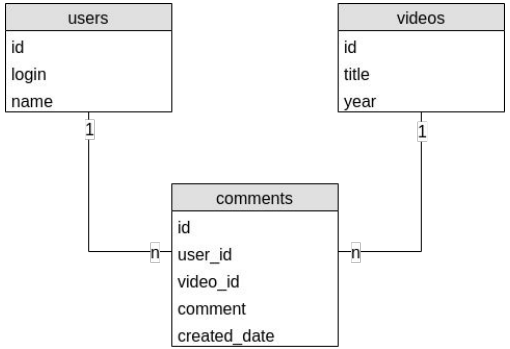
Crie novas tabelas para consultas

```
CREATE TABLE videos_by_title (
    id int,
    title text,
    runtime int,
    year int,
    PRIMARY KEY (title, id)
);


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

```
cqlsh:demo> SELECT * FROM videos;

 id | runtime | title              | year
----+---------+--------------------+------
  1 |      30 |       TDC: MongoDB | 2018
  2 |      40 | TDC: ElasticSearch | 2018
  3 |      50 |     TDC: Cassandra | 2019

(3 rows)
cqlsh:demo> SELECT * FROM videos_by_title WHERE title = 'TDC: Cassandra';

 title          | id | runtime | year
----------------+----+---------+------
 TDC: Cassandra |  3 |      50 | 2019

(1 rows)
cqlsh:demo> SELECT * FROM videos_by_year WHERE year = 2018;

 year | id | runtime | title
------+----+---------+--------------------
 2018 |  1 |      30 |       TDC: MongoDB
 2018 |  2 |      40 | TDC: ElasticSearch

(2 rows)
```

Join e Normalização

Joins e Normalização



Comentários por login de usuário

```
SELECT comment
FROM users JOIN comments ON users.id = comments.user_id
WHERE login = 'alice';
```

Comentários por título do vídeo

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

PostgreSQL

```
demo=# SELECT comment
demo-# FROM users JOIN comments ON users.id = comments.user_id
demo-# WHERE login = 'alice';
 comment
---------
 Cool!
 Good!
(2 rows)

demo=#
demo=# SELECT comment
demo-# FROM videos JOIN comments ON videos.id = comments.video_id
demo-# WHERE title = 'Interestelar';
  comment
------------
 Awesome!
 Good!
 Very good!
(3 rows)
```

Joins e Normalização

Cassandra não suporta Joins

Desnormalizar os dados

Comentários por login de usuário

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

Comentários por título do vídeo

```
CREATE TABLE comments_by_video (
    video title text,
    comment id int,
    user id int,
    video id int,
    comment text,
    PRIMARY KEY ((video_title), comment_id)
);
```

```
SELECT comment FROM comments_by_user WHERE user_login =
'alice';
```

```
cqlsh:demo> SELECT comment FROM comments_by_user WHERE user_login = 'alice';

 comment
---------
   Cool!
   Good!

(2 rows)
```

```
SELECT comment FROM comments_by_video WHERE video_title =
'Interestelar';
```

```
cqlsh:demo> SELECT comment FROM comments_by_video WHERE video_title = 'Interestelar';

 comment
-----------
   Awesome!
      Good!
 Very good!

(3 rows)
```

Resumo

Resumo

Primary Key

- Constraint unique

Tabela única por entidade

- ALLOW FILTERING
- Secondary Indexes

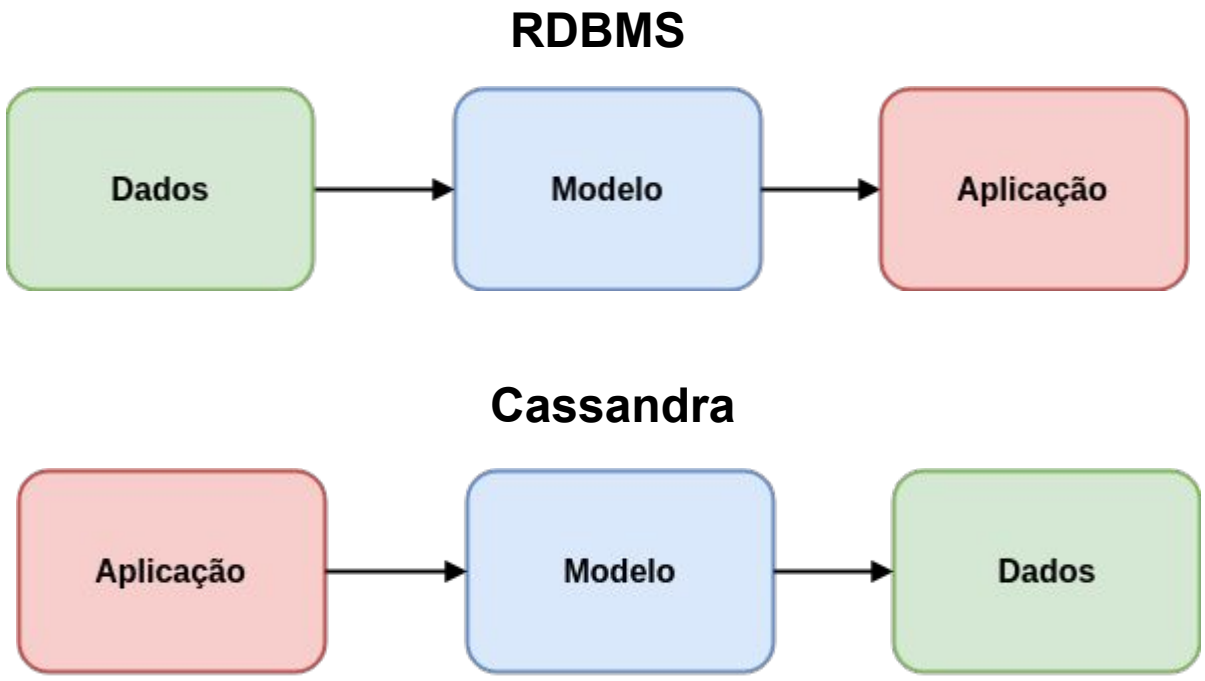
JOIN e Normalização

- Não suporta JOIN
- Desnormalizar

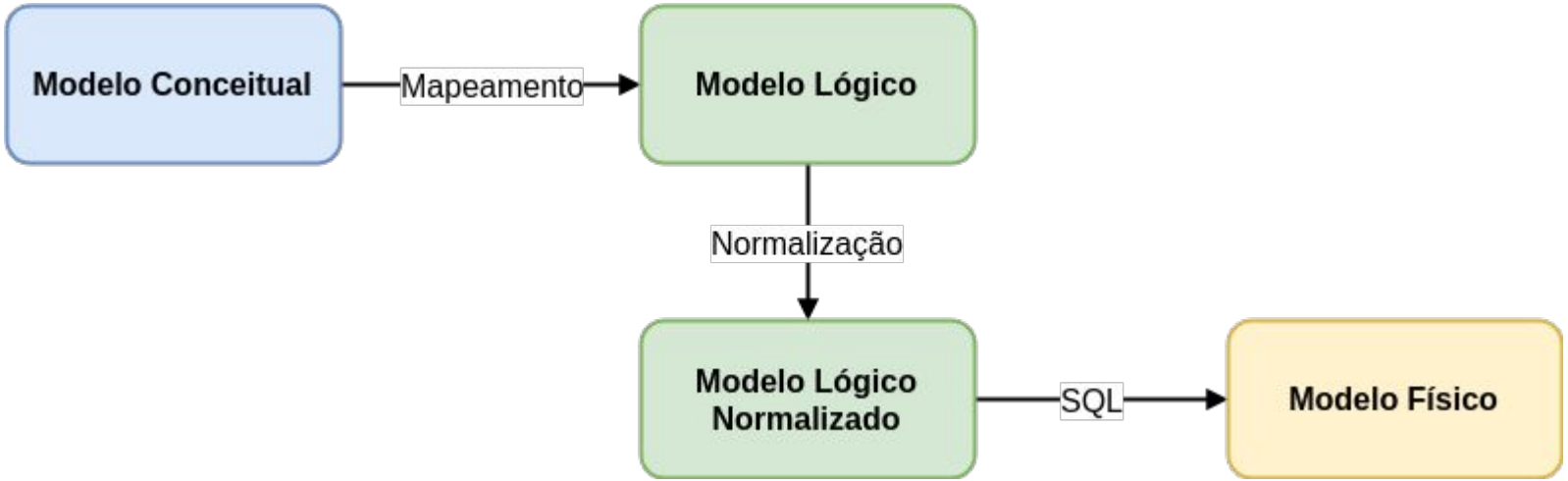
Pensar no modelo RDBMS!

Como modelar no Cassandra?

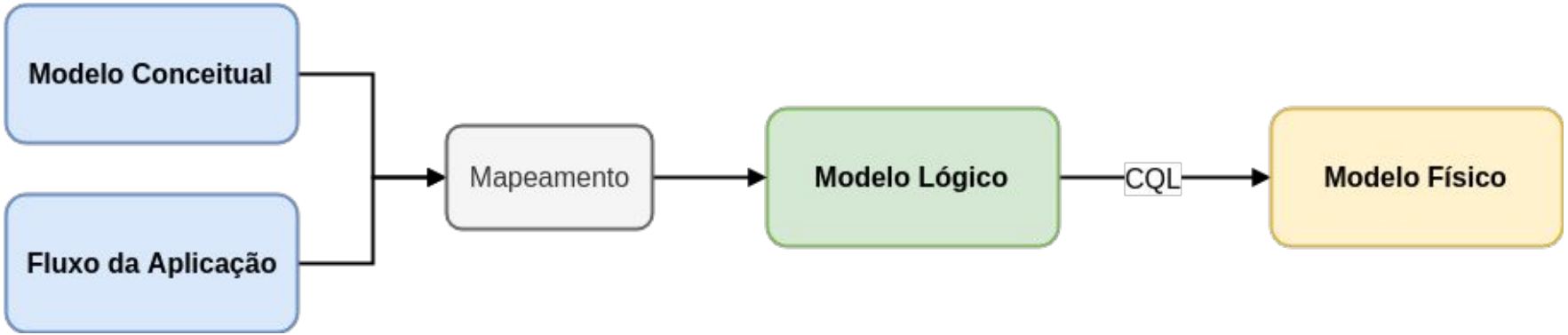
Metodologias de Modelagem de Dados



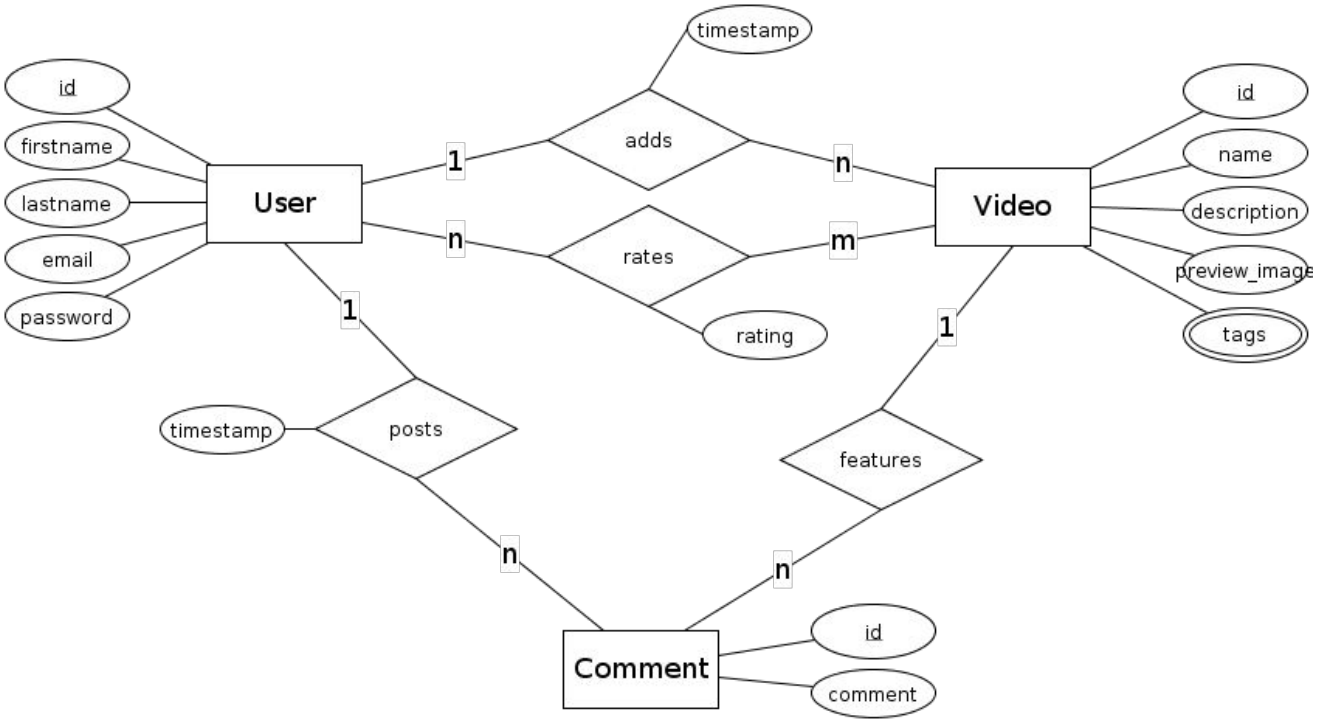
Modelagem de Dados no RDBMS



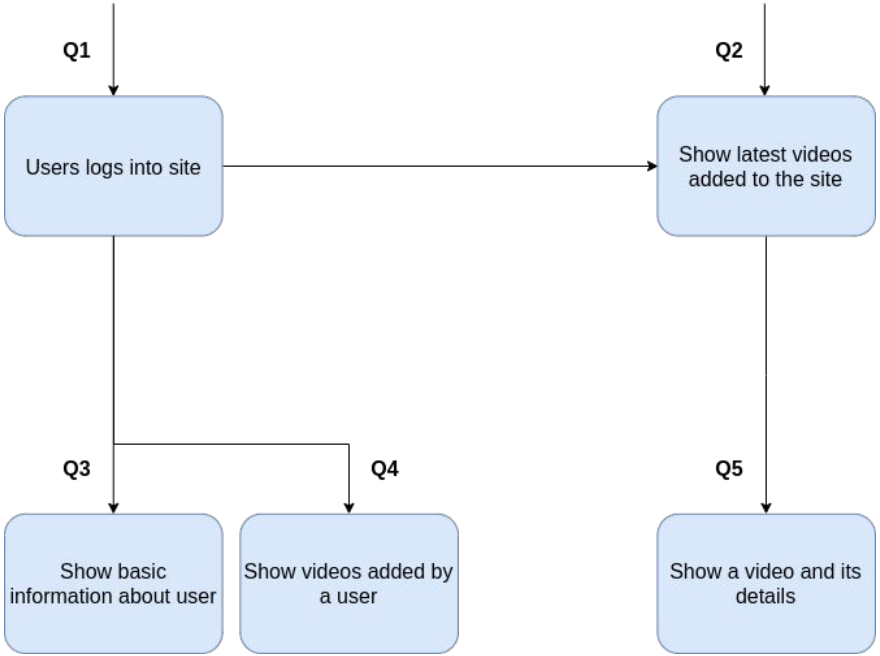
Modelagem de Dados no Cassandra



Modelo Conceitual



Fluxo da Aplicação



ACCESS PATTERNS

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

Q2: Find **mostly recently** uploaded **videos**

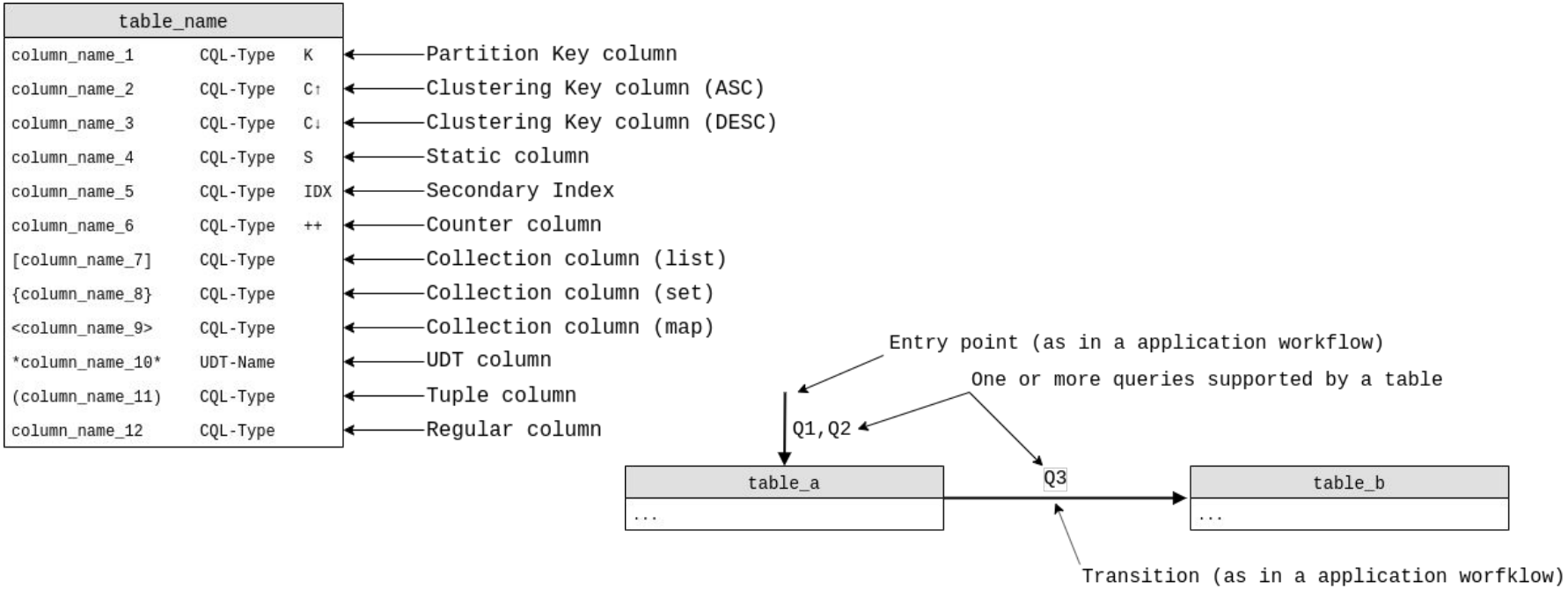
Q3: Find a **user** with a **specified id**

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

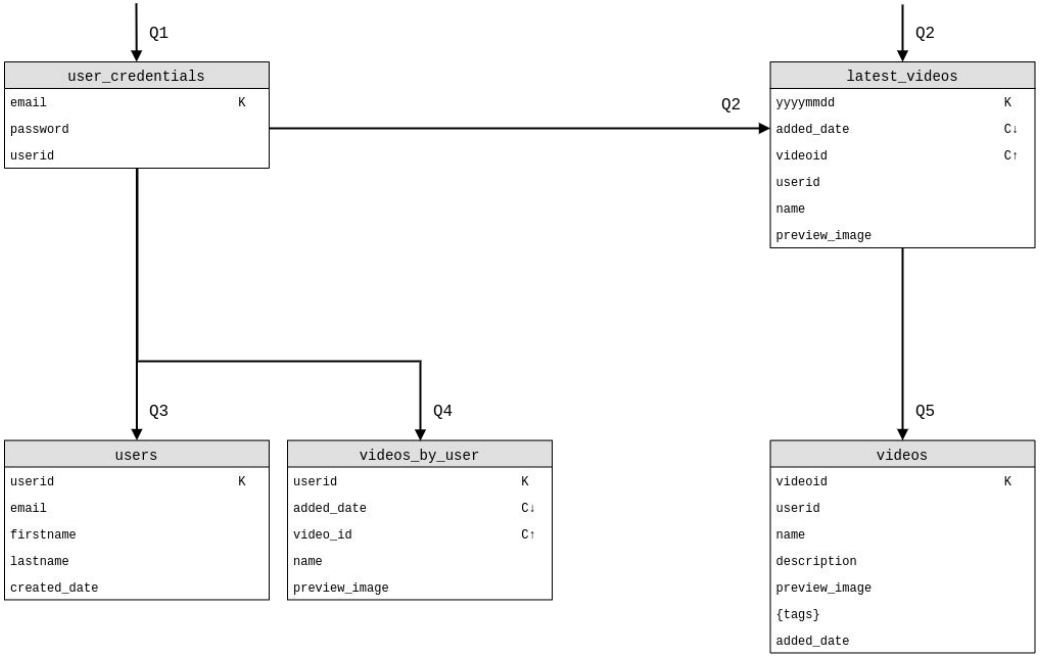
Q5: Find a **video** with **specified video id**

Mapeamento para o Modelo Lógico

Chebotko Diagram



Modelo Lógico



ACCESS PATTERNS

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

Q2: Find **mostly 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 first)

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

Modelo Físico

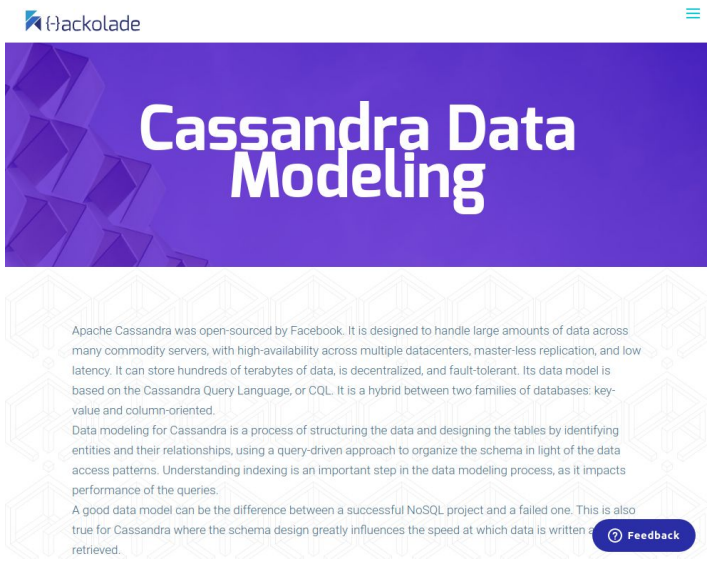
```
-- Q1: Find a user with a specified email to authenticate
CREATE TABLE user_credentials (
    email text,
    password text,
    userid uuid,
    PRIMARY KEY (email)
);
-- Q2: Find mostly recent uploaded videos
CREATE TABLE latest_videos (
    yyyymmdd text,
    added_date timestamp,
    videoid uuid,
    userid uuid,
    name text,
    preview_image text,
    PRIMARY KEY (yyyymmdd, added_date, videoid)
) WITH CLUSTERING ORDER BY (added_date DESC, videoid ASC);
-- Q3: Find a user with a specified id
CREATE TABLE users (
    userid uuid,
    firstname text,
    lastname text,
    email text,
    created_date timestamp,
    PRIMARY KEY (userid)
);
```

```
-- Q4: Find videos uploaded by a user with a known id (show most recently first)
CREATE TABLE videos_by_user (
    userid uuid,
    added_date timestamp,
    videoid uuid,
    name text,
    preview_image text,
    PRIMARY KEY (userid, added_date, videoid)
) WITH CLUSTERING ORDER BY (added_date DESC, videoid ASC);

 -- Q5: Find a video with a specified video id
 CREATE TABLE videos (
     videoid uuid,
     userid uuid,
     name text,
     description text,
     preview_image text,
     tags set<text>,
     added_date timestamp,
     PRIMARY KEY (videoid)
 );
```

Ferramentas de Modelagem

KDM Tool - kdm.dataview.org (free)

Hackolade - hackolade.com (paid)



Conclusão

Conclusão

RDBMS tem problemas de escalabilidade

Cuidado ao escolher uma nova tecnologia

Principais equívocos

- Pensar no modelo RDBMS

Modelagem do Cassandra

- Pense primeiro nas consultas da aplicação

Referências

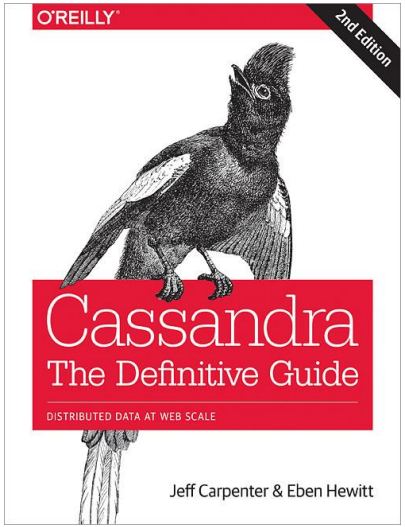
DATASTAX ACADEMY

https://academy.datastax.com/

KillrVideo

http://www.killrvideo.com/

https://killrvideo.github.io/



“With big data comes big responsibilities”

LinkedIn

@henrique--menezes

Tempest Security Intelligence

TEMPEST
Protegendo negócios
no mundo digital.