

BD2_C1A

CASSANDRA

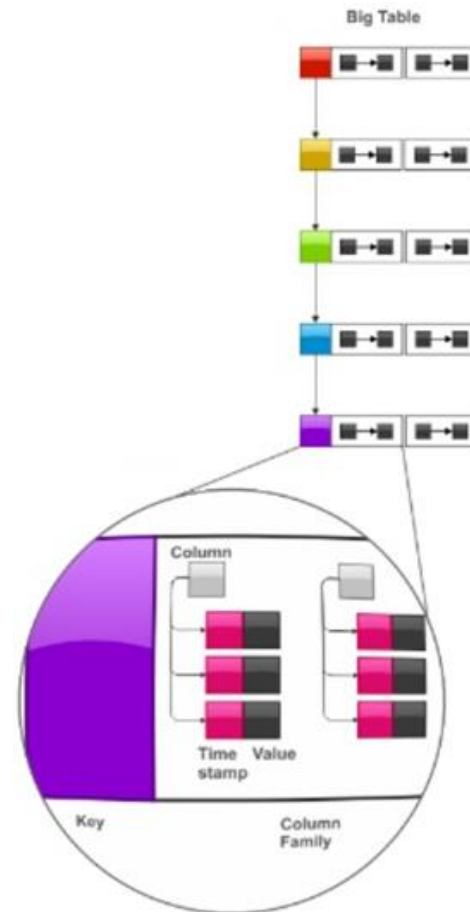
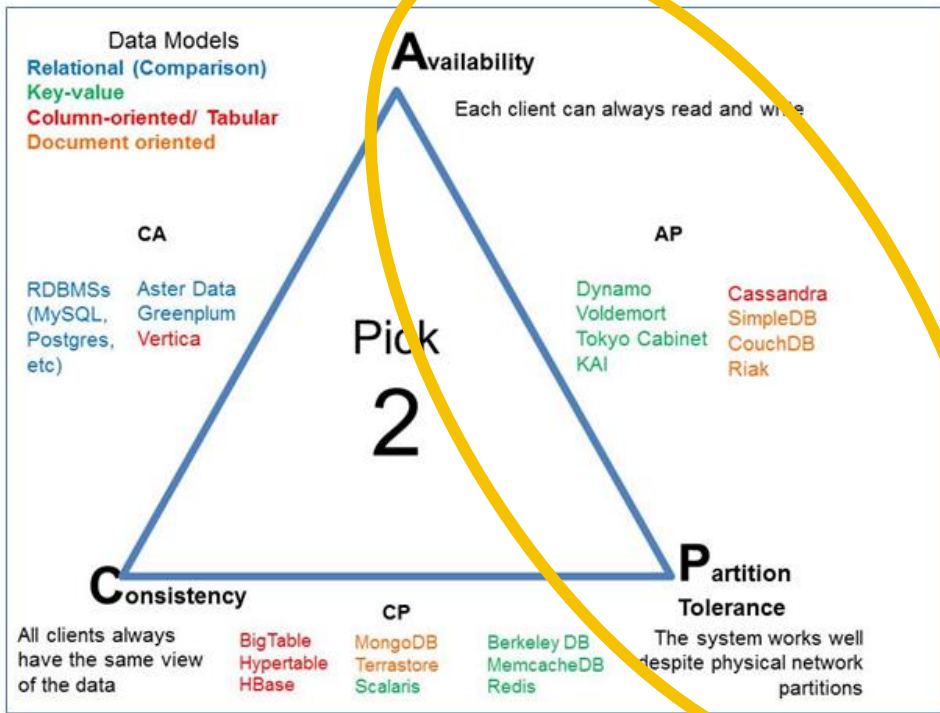
MODELADO

SERGIO ÁLVAREZ

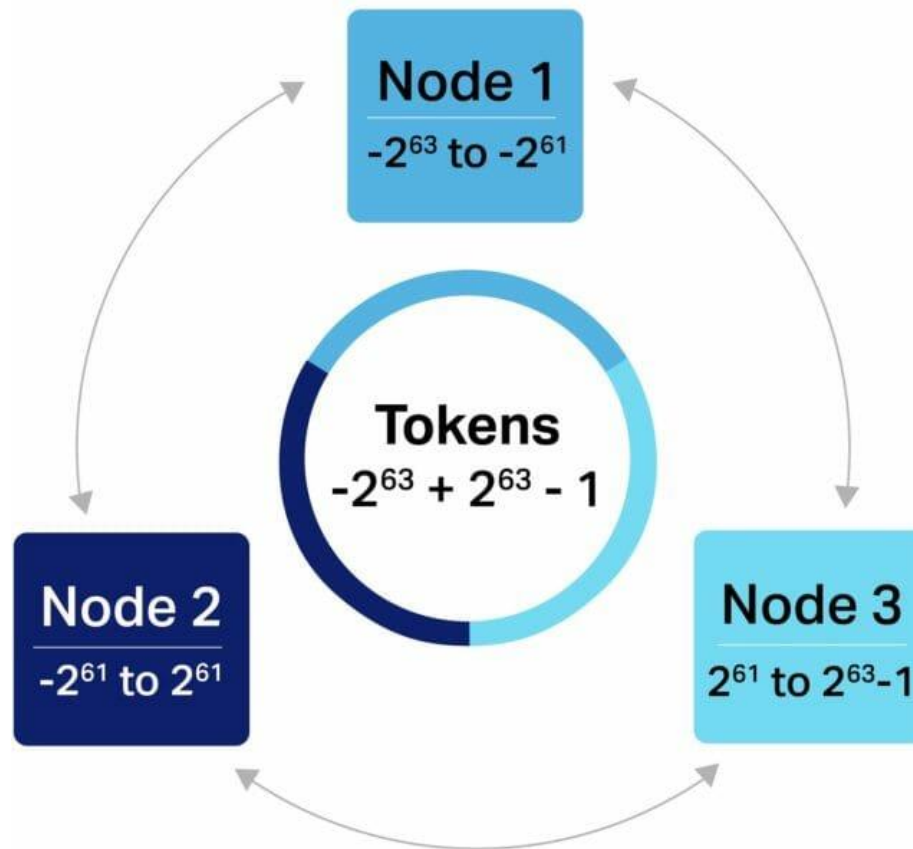
VERSIÓN 1.1

ALTA DISPONIBILIDAD

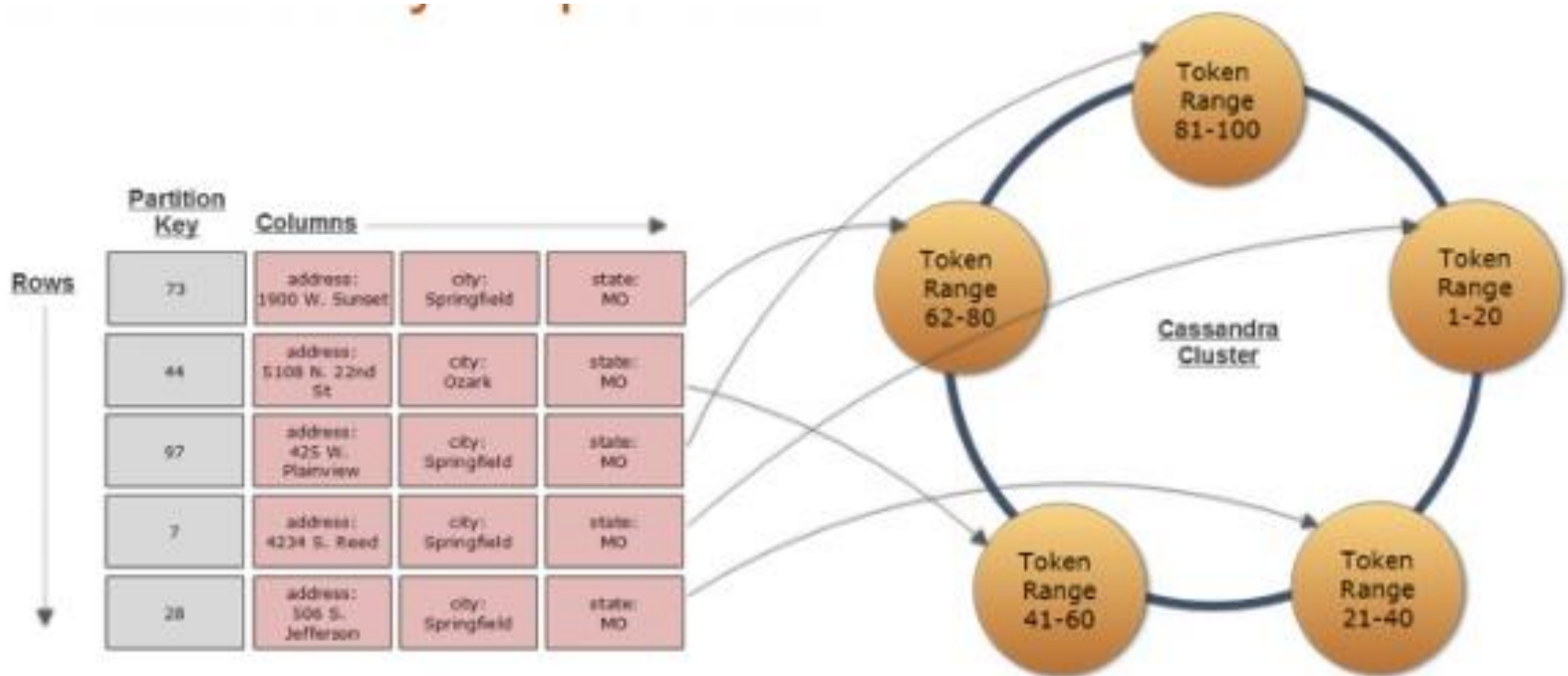
BIG TABLE



CASSANDRA USES 'TOKENS' (A LONG VALUE OUT OF RANGE -2^{63} TO $+2^{63} - 1$) FOR DATA DISTRIBUTION AND INDEXING



EJEMPLO DE DISTRIBUCIÓN INFO EN EL CLUSTER



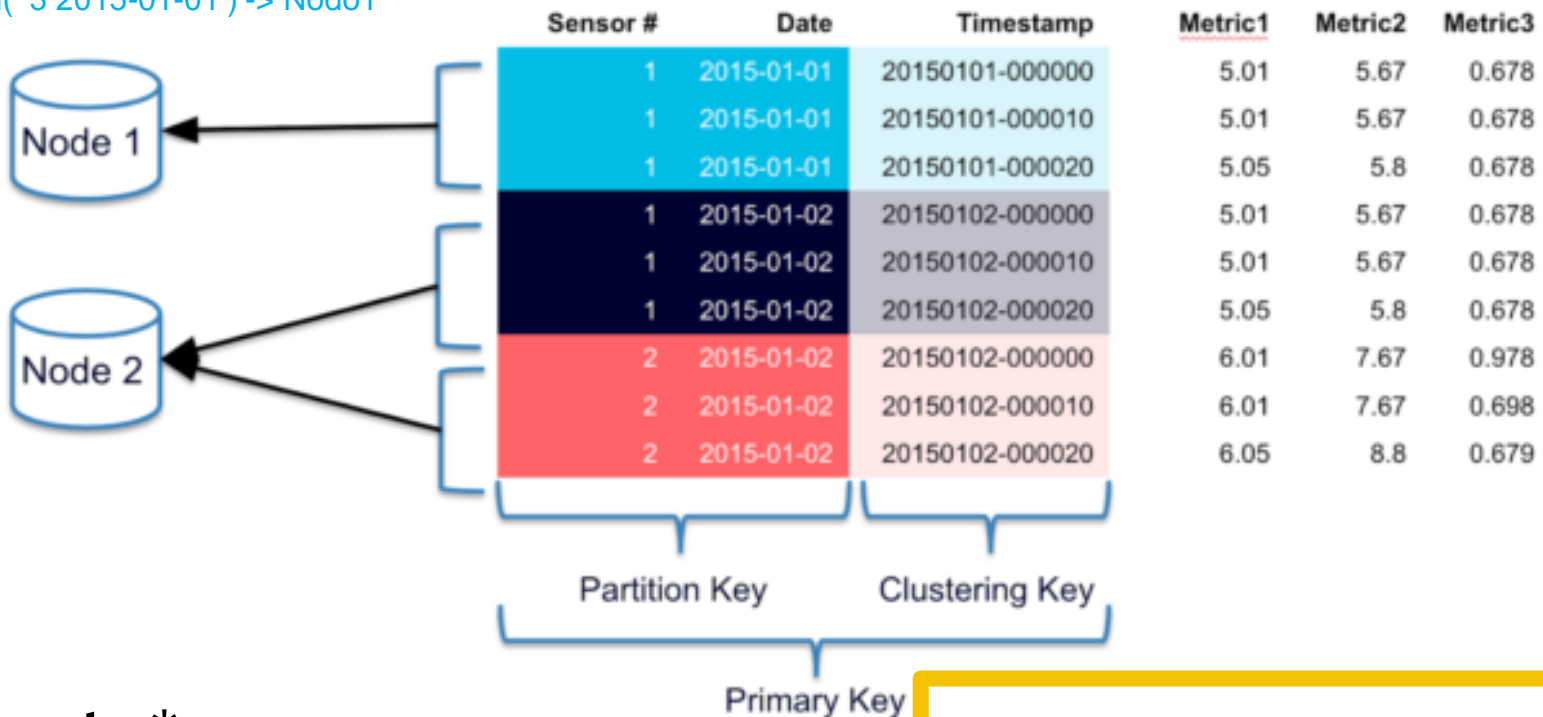
PRIMARY KEY = PARTITION KEY + [CLUSTERING COLUMNS]

HASH('1 2015-01-01') -> Nodo1

HASH('1 2015-01-02') -> Nodo2

HASH('2 2015-01-02') -> Nodo2

HASH('3 2015-01-01') -> Nodo1



```
select *  
from t  
where sensor = 3 and date = '2015-01-01'  
and Timestamp >= '20150101-000010';
```

PRIMARY KEY ((Sensor,Date),Timestamp)

CLAVE PRIMARIA

SOLO CLAVE DE PARTICIÓN

```
CREATE TABLE t (  
  id int,  
  k int,  
  v text,  
  PRIMARY KEY (id)  
);
```

```
select *  
from t  
where id = 123;
```

~~select *
from t
where k = 456;~~

~~select *
from t
where v = 'abc';~~

~~select *
from t
where id > 123;~~

~~select *
from t
where id = 123 and
k = 456;~~

~~select *
from t
where id = 123
order by k;~~

~~select *
from t
where id > 123
order by k;~~

CLAVE PRIMARIA

PARTICIÓN Y CLUSTERIZACIÓN

```
CREATE TABLE t (  
  id int,  
  c text,  
  k int,  
  v text,  
  PRIMARY KEY (id,c)  
);
```

```
select *  
from t  
where id = 123;
```

```
select *  
from t  
where id = 123 and  
      c >= 'abc';
```

```
select *  
from t  
where id = 123 and  
      c = 'abc';
```

```
select *  
from t  
where id = 123  
order by c asc;
```

CLAVE PRIMARIA

VARIAS CLAVE DE PARTICIÓN Y CLUSTERIZACIÓN

```
CREATE TABLE t (
  id1 int,
  id2 int,
  c1 text,
  c2 text,
  k int,
  v text,
  PRIMARY KEY ((id1,id2),c1,c2)
);
```

id1	id2																
123	589	c1	c2	k	v	c1	c2	k	v	c1	c2	k	v	c1	c2	k	v
		abc	a	1	hola	abc	b	11	mundo	efg	a	2	cruel	efg	x	90	.
123	458	c1	c2	k	v	c1	c2	k	v	c1	c2	k	v				
		xyz	x	9		xyz	y	9		xyz	z	9					

CLAVE PRIMARIA (1/3)

VARIAS CLAVE DE PARTICIÓN Y CLUSTERIZACIÓN

```
CREATE TABLE t (
  id1 int,
  id2 int,
  c1 text,
  c2 text,
  k int,
  v text,
  PRIMARY KEY ((id1,id2),c1,c2)
);
```

id1	id2																
123	589	c1	c2	k	v	c1	c2	k	v	c1	c2	k	v	c1	c2	k	v
		abc	a	1	hola	abc	b	11	mundo	efg	a	2	cruel	efg	x	90	.
123	458	c1	c2	k	v	c1	c2	k	v	c1	c2	k	v				
		xyz	x	9		xyz	y	9		xyz	z	9					

```
select *
from t
where id1 = 123 and
      id2 = 589;
```

```
select *
from t
where id1 = 123 and
      id2 = 589 and
      c1 > 'abc';
```

```
select *
from t
where id1 = 123 and
      id2 = 589 and
      c1 = 'abc';
```

```
select *
from t
where id1 = 123 and
      id2 = 589 and
      c1 > 'abc'
order by c2;
```

OK

CLAVE PRIMARIA (2/3)

VARIAS CLAVE DE PARTICIÓN Y CLUSTERIZACIÓN

```
CREATE TABLE t (
  id1 int,
  id2 int,
  c1 text,
  c2 text,
  k int,
  v text,
  PRIMARY KEY ((id1,id2),c1,c2)
);
```

id1	id2																
123	589	c1	c2	k	v	c1	c2	k	v	c1	c2	k	v	c1	c2	k	v
		abc	a	1	hola	abc	b	11	mundo	efg	a	2	cruel	efg	x	90	.
123	458	c1	c2	k	v	c1	c2	k	v	c1	c2	k	v				
		xyz	x	9		xyz	y	9		xyz	z	9					

```
select *
from t
where id1 = 123 and
      id2 >= 589;
```

```
select *
from t
where id1 = 123 and
      id2 = 589 and
order by c2;
```

```
select *
from t
where id1 = 123 and
      id2 = 589 and
      c2 = 'a';
```

```
select *
from t
where id1 = 123 and
      id2 = 589 and
      c1 > 'abc' and
      c2 >= 'b';
```

ERROR!

CLAVE PRIMARIA (3/3)

VARIAS CLAVE DE PARTICIÓN Y CLUSTERIZACIÓN

```
CREATE TABLE t (  
  id1 int,  
  id2 int,  
  c1 text,  
  c2 text  
  k int,  
  v text,  
  PRIMARY KEY ((id1,id2),c1,c2)  
);
```

id1	id2																
123	589	c1	c2	k	v	c1	c2	k	v	c1	c2	k	v	c1	c2	k	v
		abc	a	1	hola	abc	b	1	mundo	efg	a	1	cruel	efg	x	1	.
123	458	c1	c2	k	v	c1	c2	k	v	c1	c2	k	v				
		xyz	x	9		xyz	y	9		xyz	z	9					

```
select *  
from t  
where id1 = 123 and  
      id2 = 589 and  
      c1 = 'abc' and  
      c2 >= 'b';
```

```
select *  
from t  
where id1 = 123 and  
      id2 = 589 and  
      c1 = 'abc' and  
      c2 >= 'b'  
order by c1,c2;
```

OK

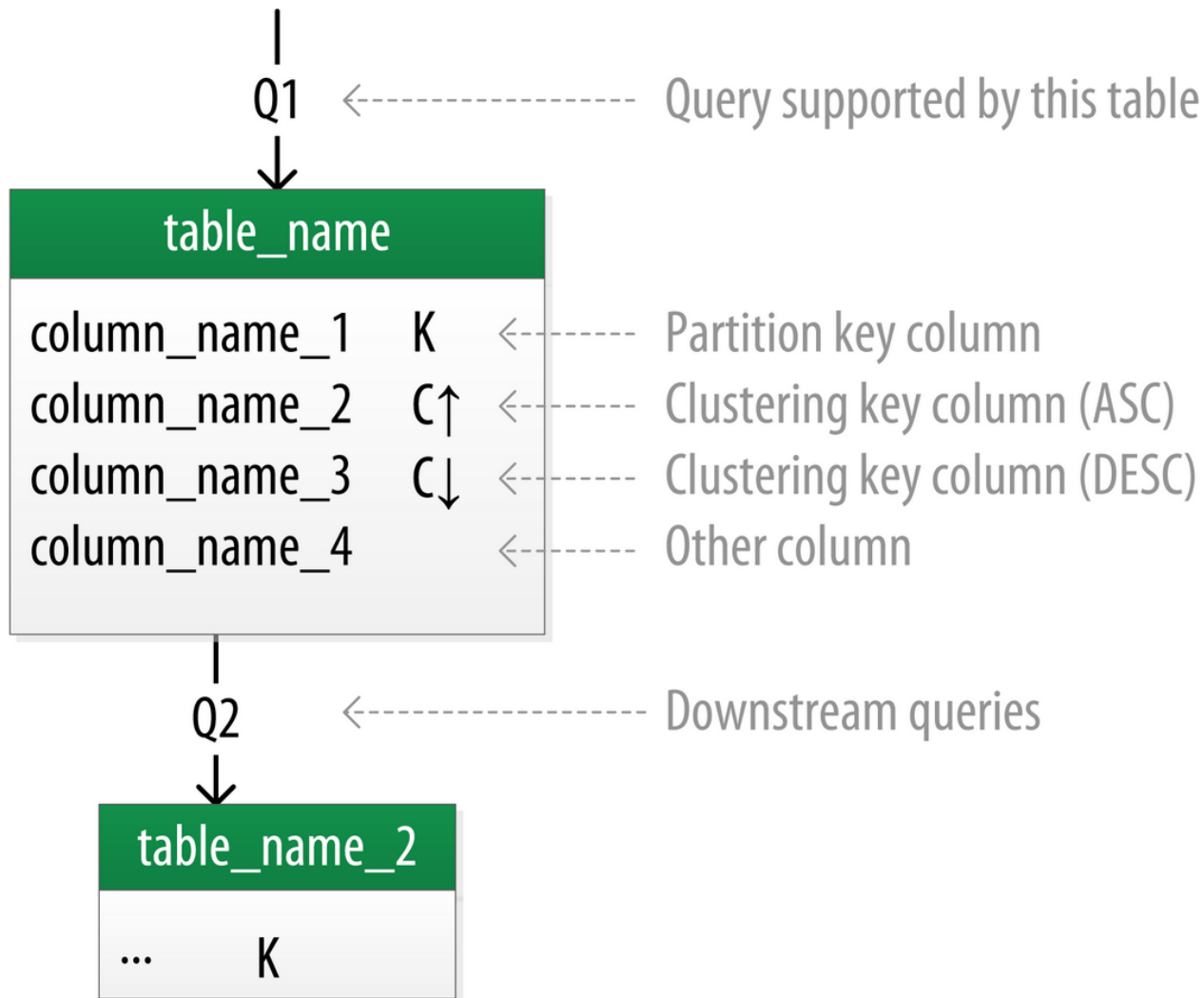
ESTANDAR

keyspace_name

table_name

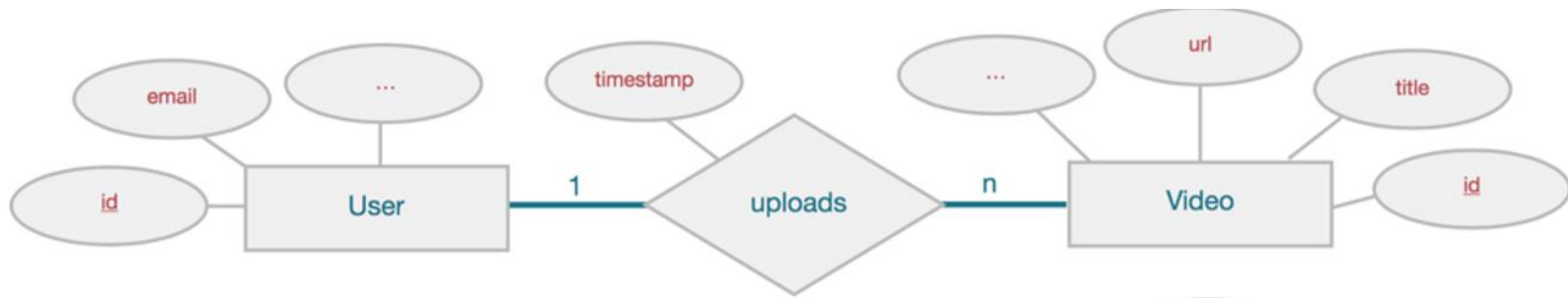
column_name_1	CQL Type	K	←-----	Partition key column
column_name_2	CQL Type	C↑	←-----	Clustering key column (ASC)
column_name_3	CQL Type	C↓	←-----	Clustering key column (DESC)
column_name_4	CQL Type	S	←-----	Static column
column_name_5	CQL Type	IDX	←-----	Secondary index column
column_name_6	CQL Type	++	←-----	Counter column
[column_name_7]	CQL Type		←-----	List collection column
{column_name_8}	CQL Type		←-----	Set collection column
<column_name_9>	CQL Type		←-----	Map collection column
column_name_10	UDT Name		←-----	UDT column
(column_name_11)	CQL Type		←-----	Tuple column
column_name_12	CQL Type		←-----	Regular column

TABLA

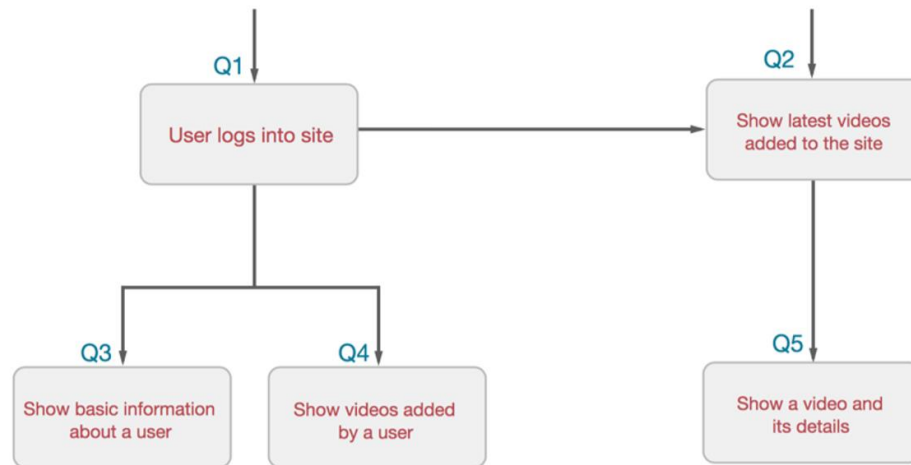


INSUMOS DISEÑO CASSANDRA

Modelo ER



WorkFlow



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** (in a time range and order by video id)

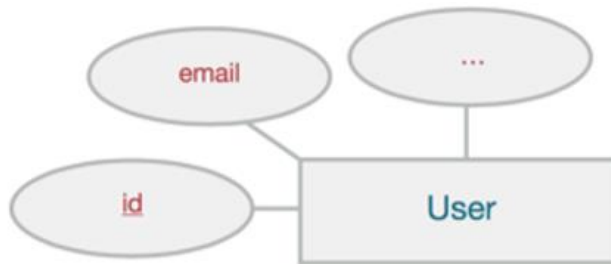
Q5: Find a video with a **specified video id**

MODELO ER CON WORKFLOW

Q1: Buscar un usuario con un email específico
Recuperar usuario id y clave

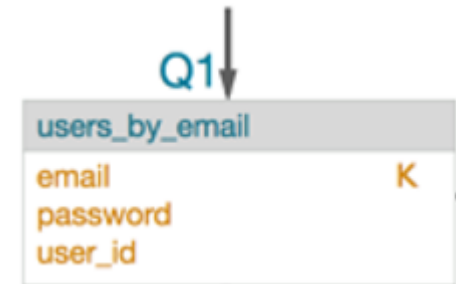
Given

email =

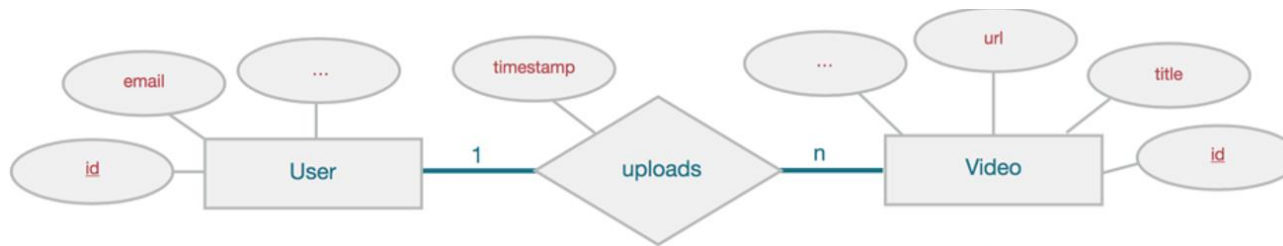


Find

Id (user_id)
password



Q4: Buscar los videos de un usuario en particular en un rango de tiempo.
Desplegando la información ordenada por el id del video.

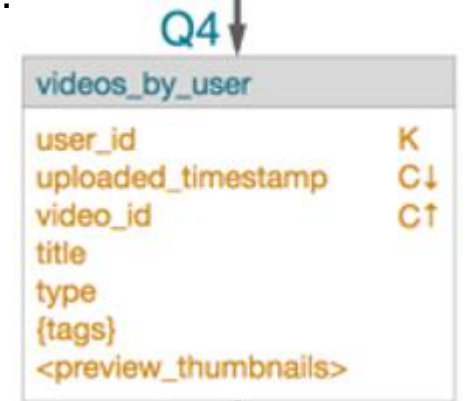


Given

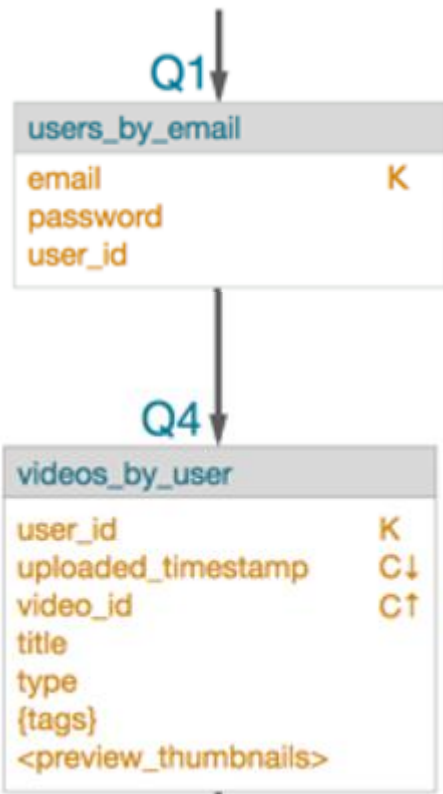
user_id (**igualdad**)
uploaded_timestamp (**rango**)

Find

video_id (**ordenar asc**)
title
type
{tags}



TRABAJANDO CON WORKFLOW



```
select password, user_id
from users_by_email
where email = 'sergalpe@gmail.com';
```

```
select title, type, tags, video_id
from videos_by_user
where
  user_id = '550e8400-e29b-41d4-a716-446655440000' and
  uploaded_timestamp <= 1597257136000
order by video_id;
```


COLUMNAS TIPO COUNTER (1)

Si se trabaja con el ejemplo de la agencia inmobiliaria. Y se tiene la siguiente estructura que guarda para una propiedad la fecha y hora que le dieron el like

```
CREATE TABLE LIKES_X_PROPIEDAD (  
  id_prop int,  
  fecha timestamp,  
  PRIMARY KEY( id_prop, fecha ));
```

id_prop							
123	<table><tr><td>fecha</td><td>2020/01/5 12:31</td><td>fecha</td><td>2020/01/5 18:01</td><td>fecha</td><td>2020/02/10 16:04</td></tr></table>	fecha	2020/01/5 12:31	fecha	2020/01/5 18:01	fecha	2020/02/10 16:04
fecha	2020/01/5 12:31	fecha	2020/01/5 18:01	fecha	2020/02/10 16:04		
587	<table><tr><td>fecha</td><td>2020/02/1 23:11</td><td>fecha</td><td>2020/02/1 14:00</td></tr></table>	fecha	2020/02/1 23:11	fecha	2020/02/1 14:00		
fecha	2020/02/1 23:11	fecha	2020/02/1 14:00				

Q1: dada una propiedad y un rango de fechas listar las fecha que le dieron like

- Given:
id_propiedad =
fecha >=
- Find:
fecha



```
SELECT fecha  
FROM LIKES_X_PROPIEDAD  
WHERE id_prop = ? AND  
      fecha >= ? AND  
      fecha <= ?;
```

Q2: dada una propiedad calcular el total de likes

- Given:
id_propiedad =
- Find:
Contar registros



```
SELECT count( * )  
FROM LIKES_X_PROPIEDAD  
WHERE id_prop = ?;
```

Q3: dada una propiedad calcular el total de likes para un día en especial

- Tip: where fecha >= 'YYYY/MM/DD 00:00' hasta fecha < 'YYYY/MM/DD 24:00'

Q4: dada una propiedad calcular el total de likes por día por un rango de fechas

COLUMNAS TIPO COUNTER (2)

Para resolver Q4 se necesita otra tabla

```
CREATE TABLE LIKES_X_DIA_PROPIEDAD (  
  id_prop int,  
  fecha_dia date,  
  total counter,  
  PRIMARY KEY( id_prop, fecha_dia ));
```

id_prop					
123		fecha	2020/01/5	fecha	2020/02/10
		total	2	total	1
587		fecha	2020/02/1	fecha	2020/02/1
		total	1	total	1

Q4: dada una propiedad y un rango de fechas listar las fecha y el número total likes por día.

- Given:
id_propiedad =
fecha >=
- Find:
fecha
total



```
SELECT fecha, total  
FROM LIKES_X_DIA_PROPIEDAD  
WHERE id_prop = ? and  
      fecha >= ? and  
      fecha <= ?;
```

Q5: dada una propiedad calcular el total de likes (igual que Q2)

- Given:
id_propiedad =
- Find:
Contar registros



```
SELECT count( * )  
FROM LIKES_X_PROPIEDAD  
WHERE id_prop = ?;
```

Q6: dada una propiedad calcular el total de likes por meses

- Tip: Necesita nueva tabla?.....

TIPS FINALES (1)

- **Todas las tablas deben tener al menos una clave tipo K.**
 - Si no tiene, se crea una columna que se llena con valor constante (Es decir siempre el mismo, por ejemplo valor 1)

```
CREATE TABLE NAVEGANTES_X_DIA (  
  id_dummy int,  
  fecha timestamp,  
  direccion_ip inet,  
  PRIMARY KEY( id_dummy, fecha, direccion_ip ));  
  
INSERT INTO NAVEGANTES_X_DIA( id_dummy, fecha, direccion_ip)  
VALUES ( 1, '2021/02/03 18:41:05.25', '192.168.0.1');
```

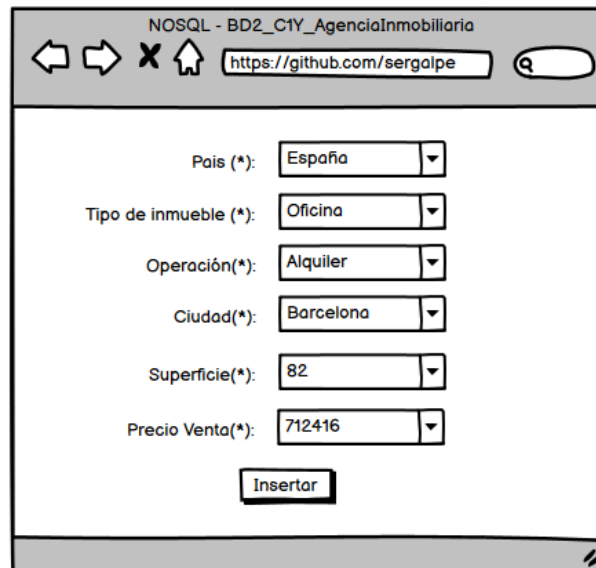


- **Apache cassandra No soporta:**
 - Joins entre dos tablas. Es decir solo se puede consultar una sola tabla.
 - Agrupaciones (group by y menos having)
- **La tablas con columnas tipo counter ++ no permiten otros tipos de columnas normales. Solo la clave primaria (K,C)**

```
UPDATE LIKES_X_DIA_PROPIEDAD  
SET total = total + 1  
WHERE id_prop = valor1  
AND fecha_dia = valor2;
```

TIPS FINALES (2)

- En muchos enunciados a problemas se describe como el usuario inserta la información. Esta información solo sirve para el modelo entidad relación.
- Ejemplo: En el ejemplo escenario de la inmobiliaria
 - La secretaria llena la información de anunciante y el sistema le asigna la fecha y el vendedor de forma automática.



The screenshot shows a web browser window with the title "NOSQL - BD2_C1Y_AgenciaInmobiliaria". The address bar contains "https://github.com/sergalpe". The form has the following fields:

- Pais (*):
- Tipo de inmueble (*):
- Operación(*):
- Ciudad(*):
- Superficie(*):
- Precio Venta(*):

Below the fields is a button labeled "Insertar".

FIN

sergalpe@gmail.com

