



Intro to Cassandra

eSolutions Grup

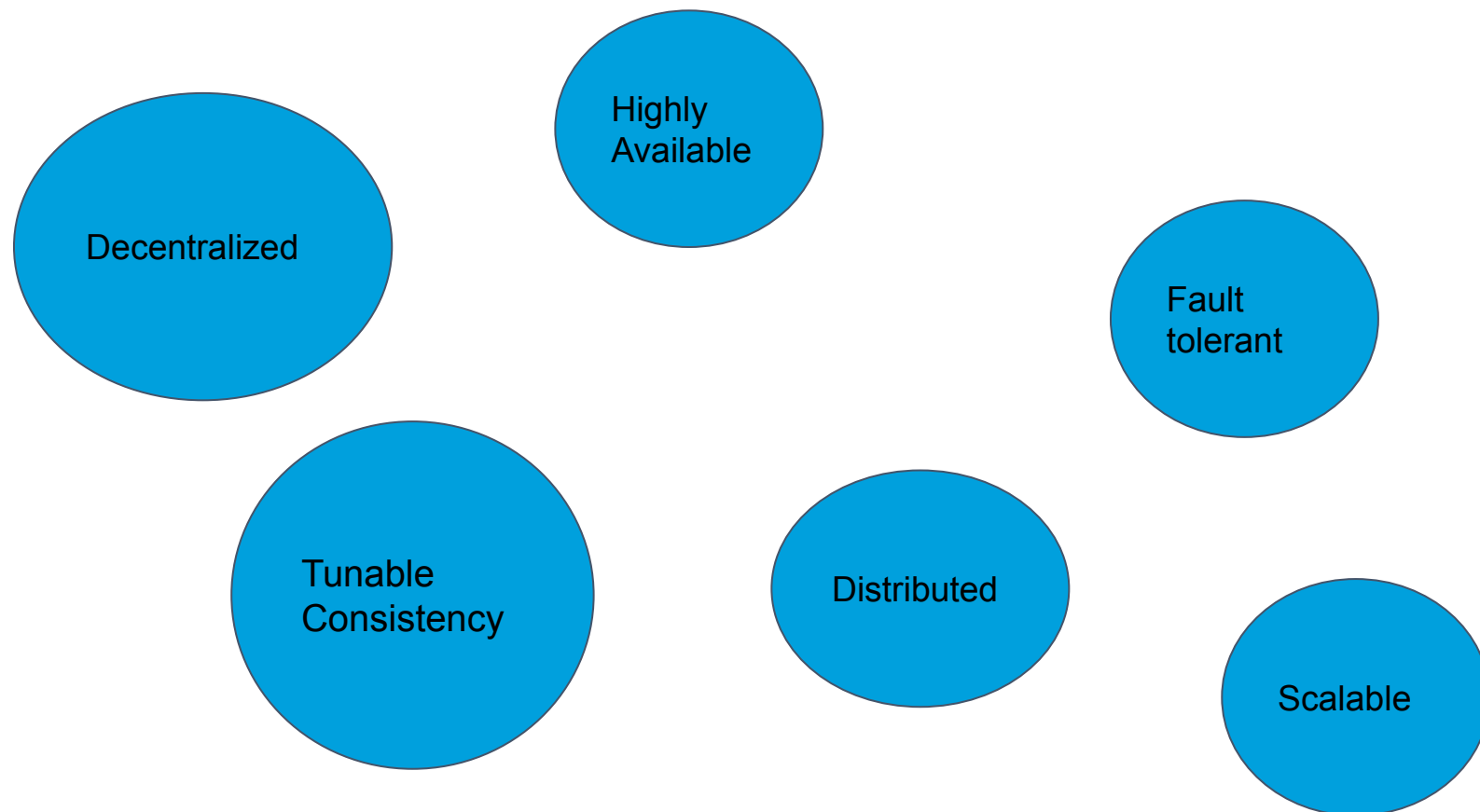
Bibiloiu Viorel
Big Data Engineer



Advantages



enable e.volution



SQL vs NoSQL



Relational (normalized)

Data tables -> Models -> Application

- + Simple read, Data Integrity
- Slow read, Complex Query



SQL vs NoSQL



NoSql (denormalized) - No JOIN!!

1 Query = 1 table

Application (customer first!) -> Models -> Data tables

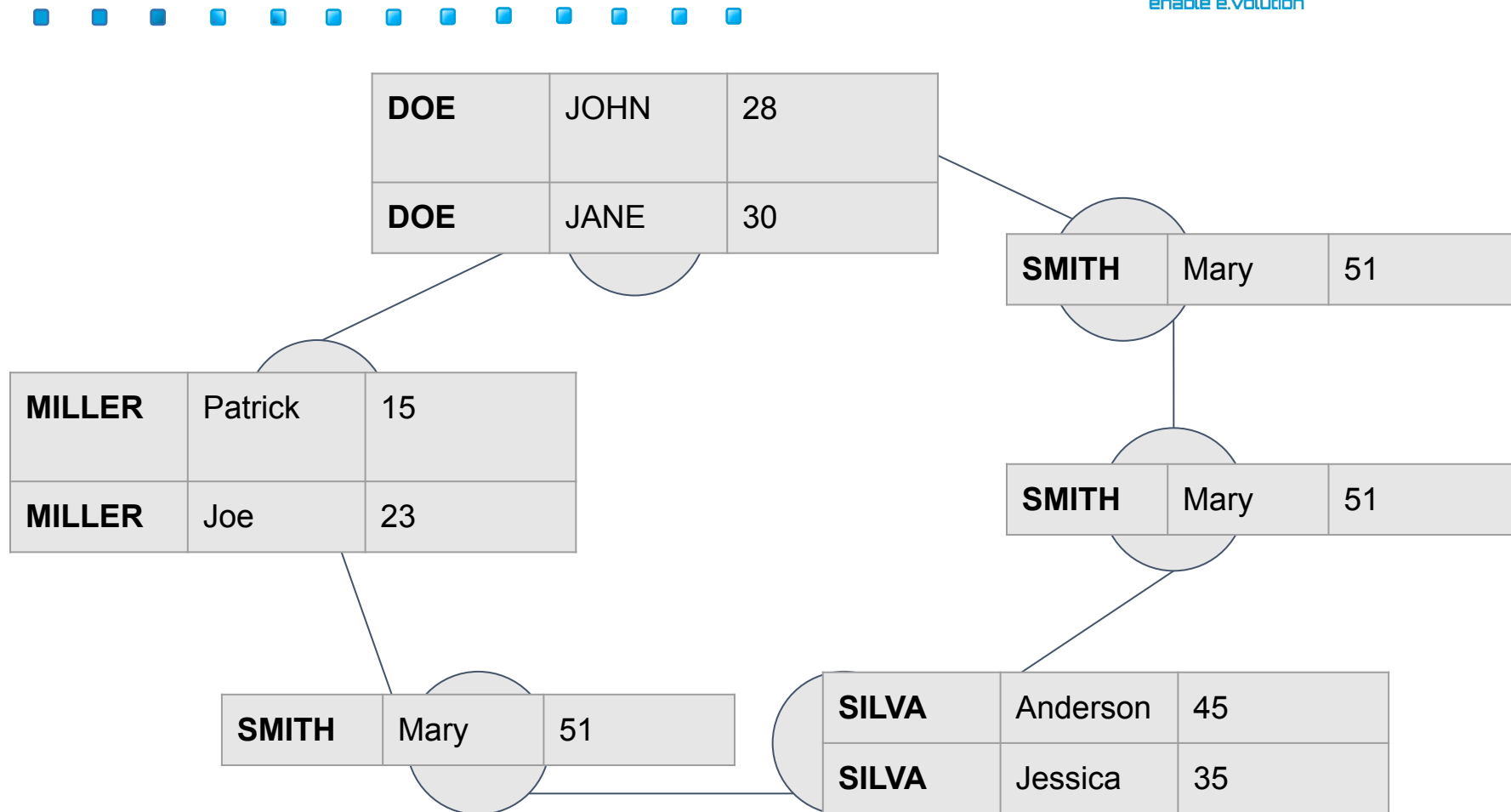
- + Quick read, Simple Query
- Multiple Writes, Manual Integrity



Partitioned data across cluster



enable e.volution



Schema details



Keyspace

Create / Alter / Drop / Use / Describe

Table

Create / Alter / Drop / Truncate / Describe

Columns

Name, Value, Timestamp, TTL

Operations: insert, update, delete

Keyspace -> Table -> Partition -> Row



Data structure



enable e.volution



Overall

Tables are rows and columns

Related rows are partitions stored on the same node(nodes)

Each row has a partition key (≥ 1 column that are hashed to determine which node stores that data)

users_by_city

| City | Last Name | First Name | Address | Email |
|-----------|-----------|------------|-------------|-------------------|
| iasi | Ionescu | Vasile | 23 First St | vasile@gmail.com |
| iasi | George | Dumitru | 2 First St | dumitru@gmail.com |
| Bucharest | Popescu | Virgil | 2 First St | virgil@gmail.com |
| Bucharest | Popescu | Virgil | 2 First St | virgil2@gmail.com |



Table creation

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

```
CREATE TABLE bdwvideo.users_by_city(  
    city text,  
    last_name text,  
    first_name text,  
    address text,  
    email text,  
    PRIMARY KEY ((city), last_name, first_name, email))  
WITH CLUSTERING ORDER BY (last_name ASC, first_name ASC, email ASC);
```



Tunable Consistency



ALL: highest consistency and the lowest availability

LOCAL_QUORUM: used in multiple data center clusters and maintain consistency locally

ONE, TWO, THREE: checks closest nodes to the coordinator

ANY: provides low latency and a guarantee that a write never fails



Exercises



1. `docker-compose up -d`

<https://github.com/eSolutionsGrup/cassandra-intro>

```
docker exec -ti bdw_node01_1 cqlsh
```

```
CREATE KEYSPACE bdwvideo WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'} AND durable_writes = true;
```

```
use bdwvideo;
```

2. <https://astra.datastax.com>

- a. Register demo account
- b. Create database **bdwvideo** (in any Area ex West Europe)
- c. Go to Dashboard/bdwvideo then go to CQL console

TODO: create `bdwvideo.users_by_city` table
`desc tables; //` to see the tables created



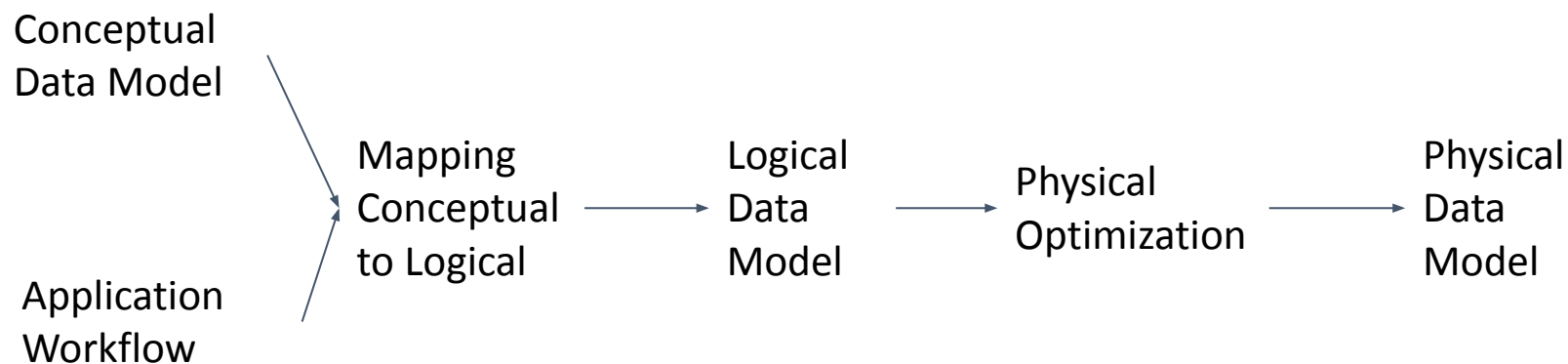
Pain Points



- No Joins
- No arbitrary where conditions (only Primary Key columns)
- No column filtering
- Mandatory Partition Key



Data Modeling: Schematics



Video platform bdwvideo



Entity-Relationship diagram

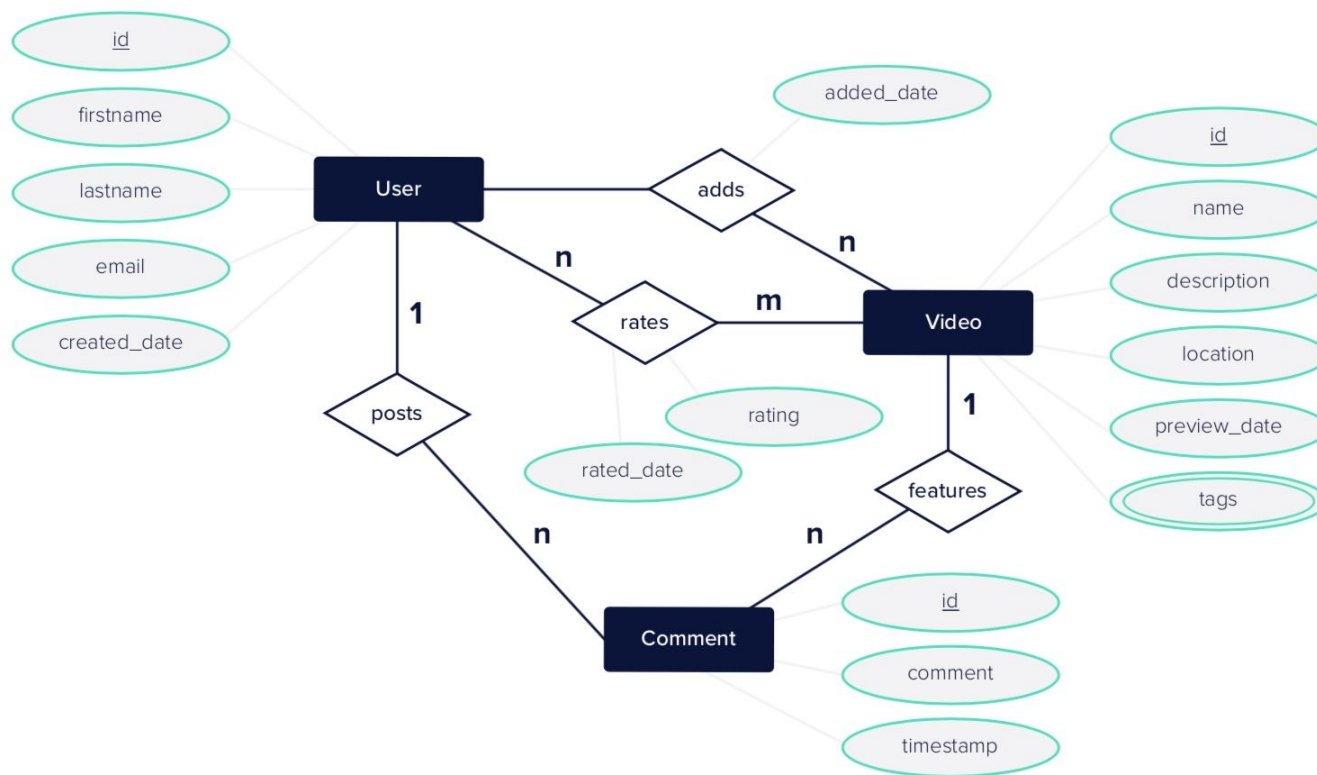


Fig 1

<https://www.datastax.com/blog/five-steps-awesome-data-model>



Application workflow



enable e.volution

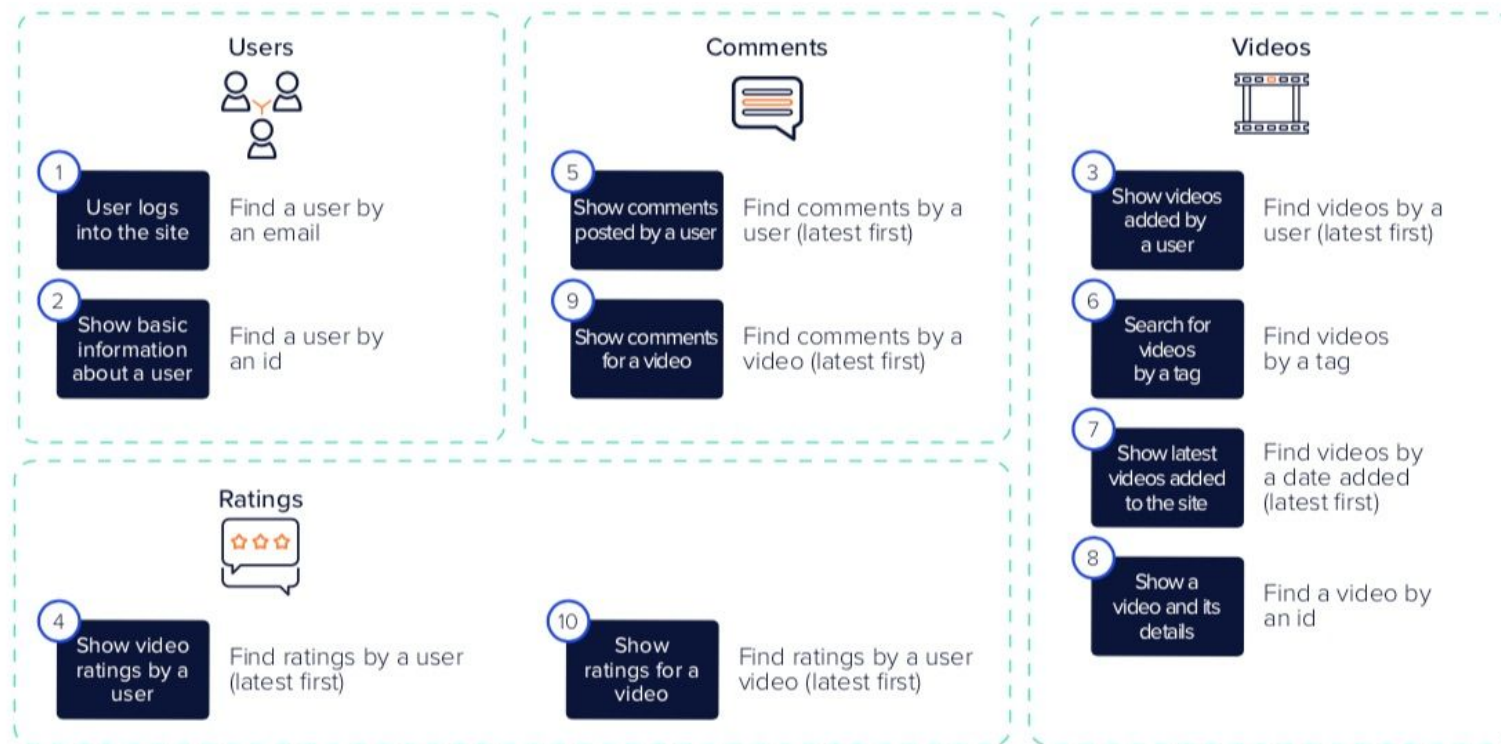


Fig 2

<https://www.datastax.com/blog/five-steps-awesome-data-model>



Access patterns (Users)



User by email (login)?

```
create table user_credentials(  
    email text,  
    password text,  
    userid uuid,  
    primary key (email)  
);
```

User by id (user profile)?

```
create table users(  
    userid uuid,  
    firstname text,  
    lastname text,  
    email text,  
    created_date timestamp,  
    primary key (userid)  
);
```



enable e.volution



Access patterns inserts(Users)



enable e.volution



```
insert INTO users (userid , firstname , lastname , email , created_date) VALUES (  
uuid(),'john','doe','john@doe', '2021-10-09');
```

```
select uuid from users;
```

```
update users set lastname='Johnson' where userid = ;
```

```
select uuid from users;
```

```
update users set lastname='Johnson' where userid =  
a62485ed-f11d-485e-bd24-14b25a5ba0ff;
```



Access patterns (Video)



Video by id (for details)?

```
create table videos(  
    videoid uuid,  
    userid uuid,  
    name text,  
    description text,  
    location text,  
    location_type int,  
    preview_image_location text,  
    tags set<text>,  
    added_date timestamp,  
    primary key (videoid)  
);
```

Video by user_id (latest)

```
create table user_videos(  
    userid uuid,  
    added_date timestamp,  
    videoid uuid,  
    name text,  
    preview_image_location text,  
    primary key (userid, added_date,  
videoid)  
    ) with clustering order by (  
    added_date desc,  
    videoid asc);
```



Duplicating data problems



How frequently does the data change?

Do I have all I need to update duplicates and maintain consistency?

A:

```
BEGIN BATCH
```

```
INSERT INTO videos ...
```

```
INSERT INTO user_videos ...
```

```
APPLY BATCH;
```



Access patterns (Video latest)



enable e.volution

Latest videos ?

```
create table latest_videos(  
    video_date text,  
    added_date timestamp,  
    videoid uuid,  
    name text,  
    preview_image_location text,  
    primary key (video_date,  
    added_date, videoid))  
    with clustering order by  
    (added_date desc, videoid asc);
```

```
insert into latest_videos ( video_date ,  
    added_date , videoid , name ,  
    preview_image_location ) VALUES ( '2021-10-12',  
    toTimestamp(now()), uuid(),'best name','best  
    preview');
```

```
insert into latest_videos ( video_date ,  
    added_date , videoid , name ,  
    preview_image_location ) VALUES ( '2021-10-12',  
    toTimestamp(now()), uuid(),'best name','best  
    preview');
```

```
insert into latest_videos ( video_date ,  
    added_date , videoid , name ,  
    preview_image_location ) VALUES ( '2021-10-12',  
    toTimestamp(now()), uuid(),'best name','best  
    preview');
```



Access patterns (Video latest)



Latest videos ?

```
create table latest_videos_bucketed(  
    video_date text,  
    bucket int,  
    added_date timestamp,  
    videoid uuid,  
    name text,  
    preview_image_location text,  
    primary key ((video_date, bucket),  
        added_date, videoid))  
    with clustering order by  
    (added_date desc, videoid asc);
```

```
insert into latest_videos_bucketed (  
    video_date ,bucket, added_date , videoid ,  
    name , preview_image_location ) VALUES (  
    '2021-10-12',1, toTimestamp(now()),  
    uuid(),'best name','best preview');
```

```
insert into latest_videos_bucketed (  
    video_date ,bucket, added_date , videoid ,  
    name , preview_image_location ) VALUES (  
    '2021-10-12',1, toTimestamp(now()),  
    uuid(),'best name','best preview');
```

Query?



enable e.volution



Best Practices



Store in the same place what you want to retrieve together

Avoid Big Partitions (max 100K rows,max 100 MB)

Avoid Hot Partitions

users -> PRIMARY KEY (user_id)

comments_by_video -> PRIMARY KEY ((video_id), comment_id)

users_by_country -> PRIMARY KEY ((country), user_id)



Best Practices



Store in the same place what you want to retrieve together

Avoid Big Partitions (max 100K rows,max 100 MB)

Avoid Hot Partitions

users -> PRIMARY KEY (user_id) ✓

comments_by_video -> PRIMARY KEY ((video_id), comment_id) ✓

users_by_country -> PRIMARY KEY ((country), user_id) ❌



Good scaling model?



Load/Performance Tests

cassandra-stress (.yml)

https://cassandra.apache.org/doc/latest/cassandra/tools/cassandra_stress.html



Scalability



2 Nodes -> 100K ops/sec

4 Nodes -> 200K ops/sec

8 Nodes -> 400K ops/sec

16 Nodes -> 800K ops/sec



enable e.volution



Final thought



Thank you!



- eSolutions academy (<https://academy.esolutions.ro/>)
- Datastax Academy (<https://academy.datastax.com/>)
- Git (<https://github.com/eSolutionsGrup/cassandra-intro>)

