DATASTAX ACADEMY

DS220

02: Building Blocks of a Good Data Model

DATASTAX ACADEMY

Objectives

Over the next few sections, we will cover:

- The general and table specific Apache Cassandra™ terminology
- Supported Apache Cassandra™ data types
- The Apache Cassandra™ storage structure
- How the partitioning works
- Partition and primary keys
- Clustering columns and the data model
- Denormalization

DATASTAX ACADEMY

Fundamentals of an Apache Cassandra™ Table

DATASTAX ACADEMY

Apache Cassandra™ Terminology

Terms and definitions to get your head around

- **Data model:**
  - An abstract model for organizing elements of data
  - In Apache Cassandra™ this is based on the queries you want to perform
- **Keyspace:**
  - Similar to relational schema—outermost grouping of data
  - All tables live inside a keyspace
  - Keyspace is the container for replication
- **Table:**
  - Grouped into keyspaces
  - Contain columns
- **Partition:**
  - Row(s) of data that are stored on a particular node in your based on a partitioning strategy

Apache Cassandra™ Table Terminology

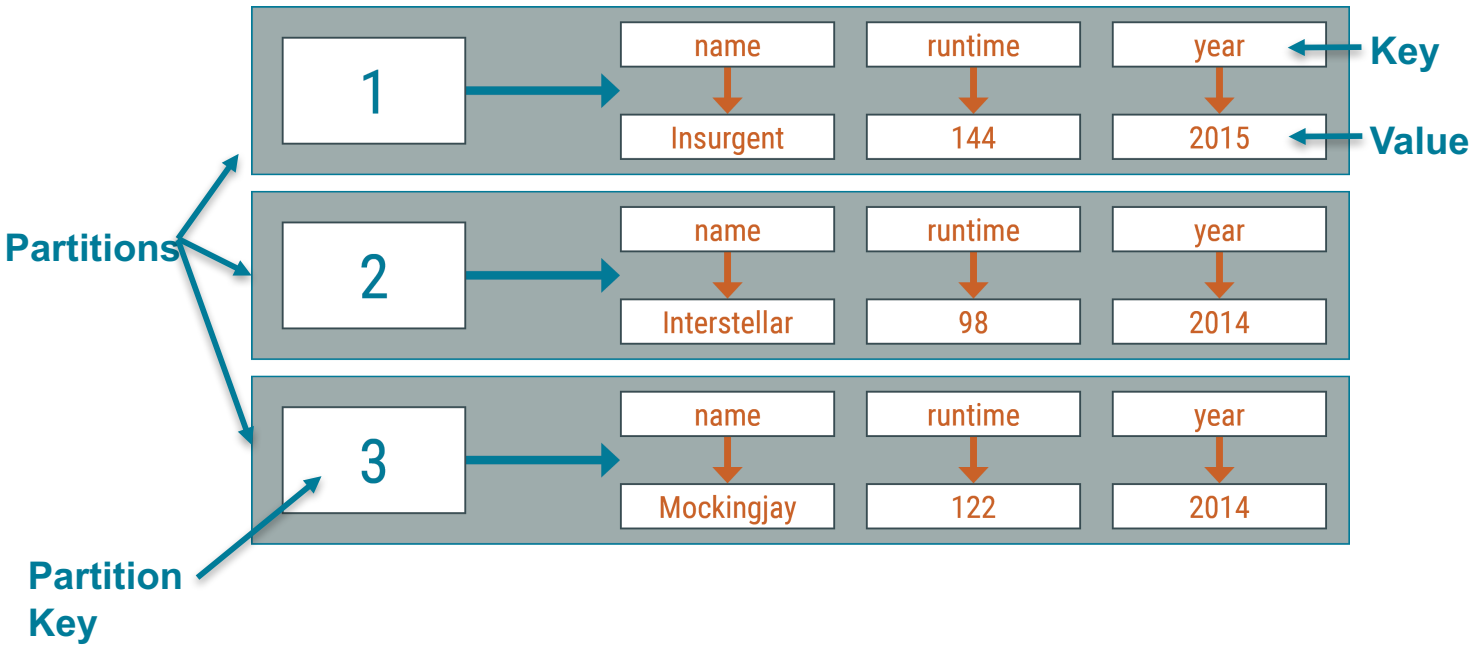
More specific to the tables themselves

- **Row**:
  - One or more CQL rows stored together on a partition
- **Column:**
  - Similar to a column in a relational database
  - **Primary key:**
  - Used to access the data in a table and guarantees uniqueness
- **Partition key:**
  - Defines the node on which the data is stored
- **Clustering column:**
  - Defines the order of rows within a partition

DATASTAX ACADEMY

Graphical Representation of Terms

A picture is worth 1000 words



Apache Cassandra™ Data Types—Text

Text data types

- **Ascii:** US-ASCII characters
- **Text:** UTF-8 encoded string
- **Varchar:** UTF-8 encoded string

Data Types—Integers

Integer data types

- **Tinyint**: 8-bit signed integer
- **Smallint**: 16-bit signed
- **Int**: 32-bit signed integer
- **Bigint**: 64-bit signed integer
- **Varint**: Arbitrary-precision integer--F-8 encoded string
- **Decimal**: Variable-precision decimal, supports integers and floats.
- **Float**: 32-bit IEEE-754 floating point
- **Double:** 64-bit IEEE-754 floating point

DATASTAX ACADEMY

Data Types—Date, Time and Unique Identifiers

Time, timestamp and unique identifiers

- **Date**: 32-bit unsigned integer--number of days since epoch (Jan 1, 1970)
- **Duration**: Signed 64-bit integer--amount of time in nanoseconds
- **Time**: Encoded 64-bit signed--number of nanoseconds since midnight
- **Timestamp**: 64-bit signed integer--date and time since epoch in milliseconds

- **UUID:** 128 bit universally unique identifier--generate with the UUID function
- **TimeUUID:** unique identifier that includes a“conflict-free” timestamp--generate with the NOW function

DATASTAX ACADEMY

Data Types—Specialty Types

Some that don't fit the other categories

- **Blob**: Arbitrary bytes (no validation), expressed as hexadecimal
- **Boolean**: Stored internally as true or *false*
- **Counter**: 64-bit signed integer--only one counter column is allowed per table
- **Inet**: IP address string in IPv4 or IPv6 format

DATASTAX ACADEMY

Partitioning and Storage Structure

DATASTAX ACADEMY

Partitions

To truly understand data modeling, you must grok partitioning

Over the next few slides, we will cover the following in depth:

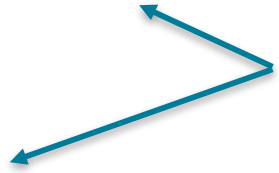
- Partitions
- Partition keys
- Composite partition keys

Queries in a relational context

In a relational world, these queries should work on videos table

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

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



These would have been possible in relational world due to joins.

Apache Cassandra™ output:

```
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: "
```

DATASTAX ACADEMY

Videos Table Example

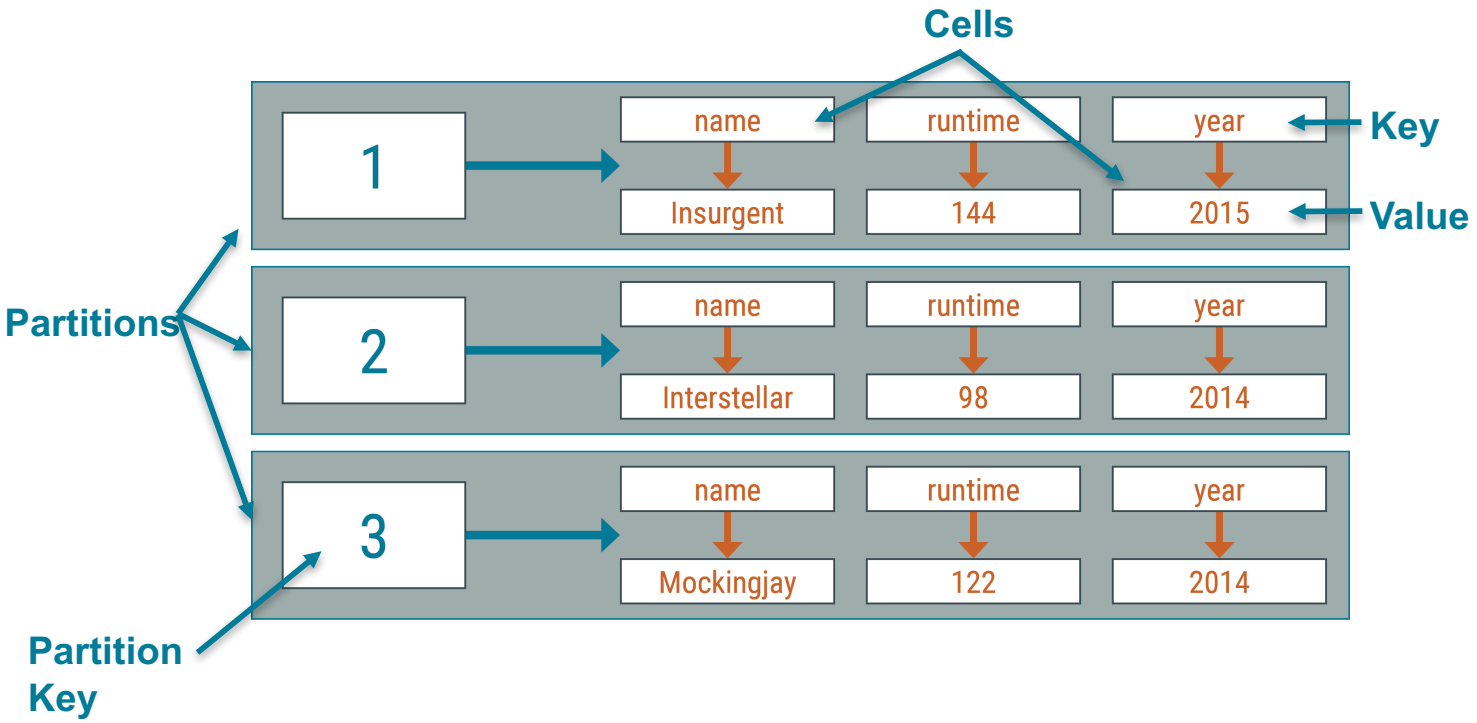
Simpler videos schema for demo purposes

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

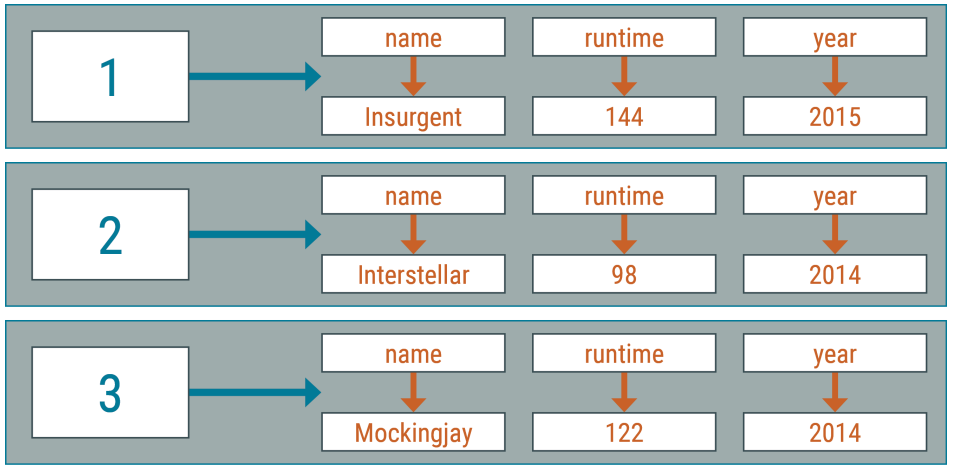
id	name	runtime	year
1	Insurgent	119	2015
2	Interstellar	98	2014
3	Mockingjay	122	2014

DATASTAX ACADEMY

Apache Cassandra™ Storage Structure



CQL Tables

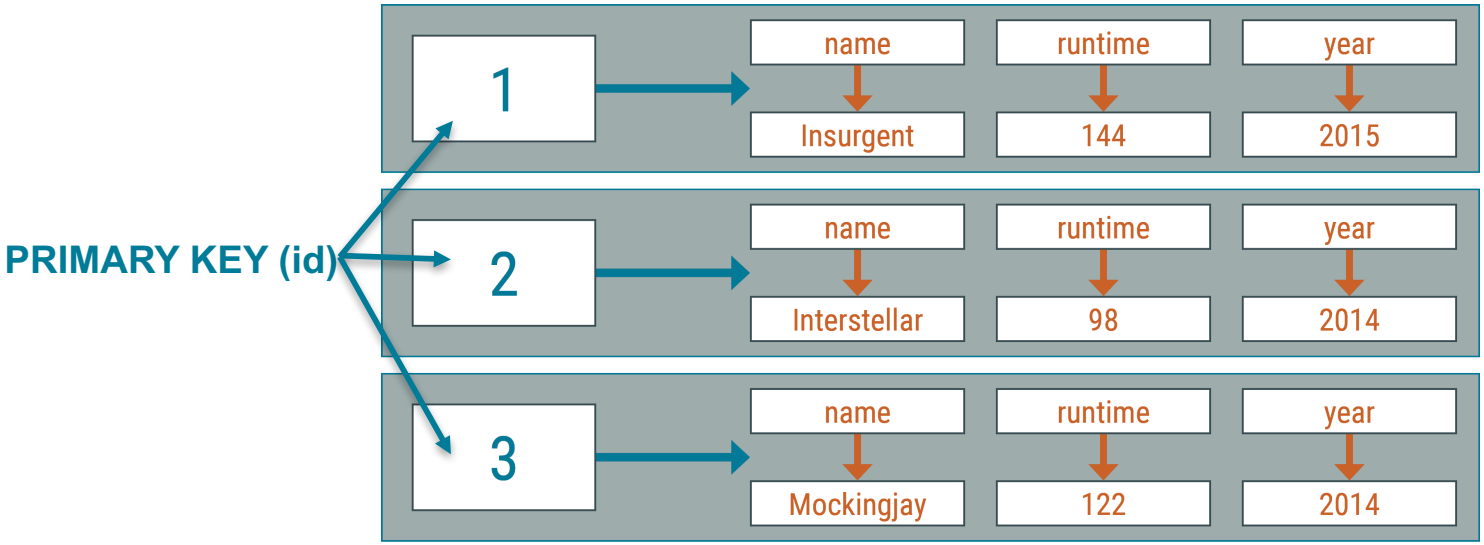



```
SELECT *
FROM videos
```

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

Determining Partition Keys

CQL's PARTITION KEY clause determines partitioning criteria in the primary key

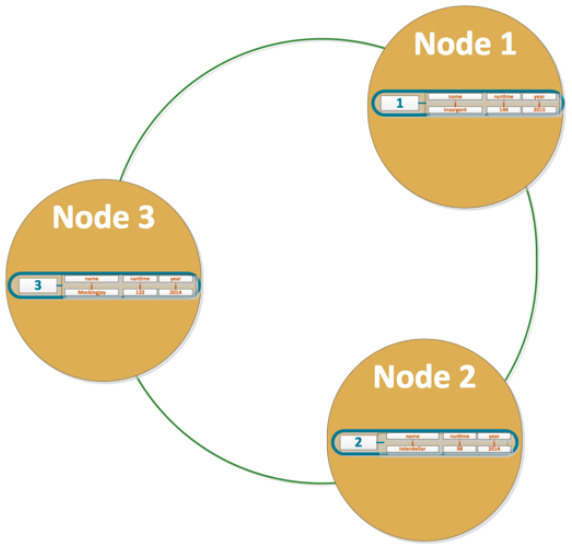


DATASTAX ACADEMY

Partition Storage

WHERE on non-primary key columns

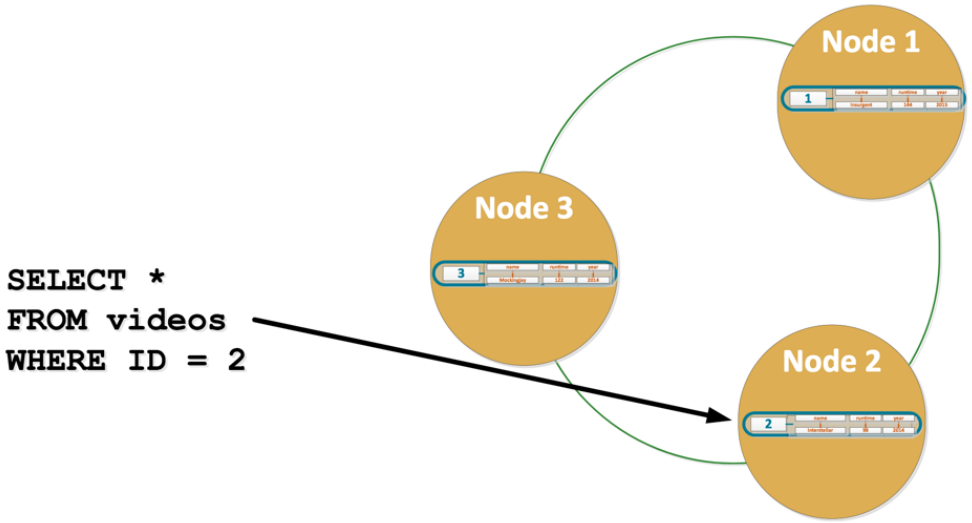
- Apache CassandraTM distributes partitions across nodes
- WHERE on any field other than partition key would require a scan of all partitions on all nodes
- Inefficient access pattern



WHERE and Partition Keys

WHERE on partition key values

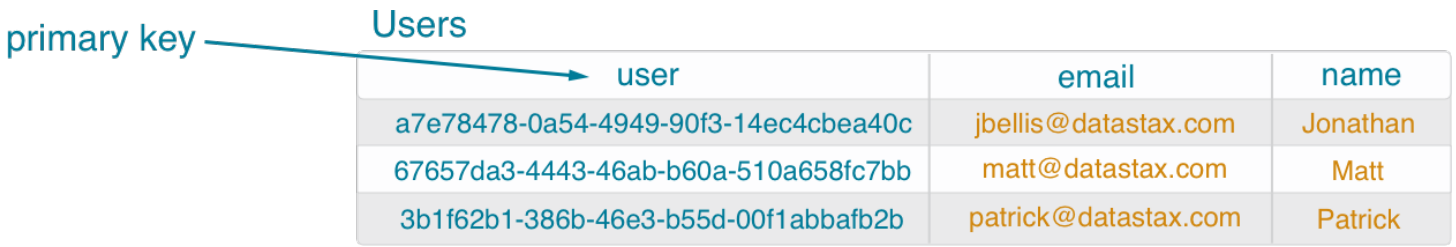
- We can WHERE on a partition key value as well as clustering columns
- Apache Cassandra™ uses a hashing algorithm to quickly determine which node(s) contain the desired partition



Primary Key

Simple Primary Key

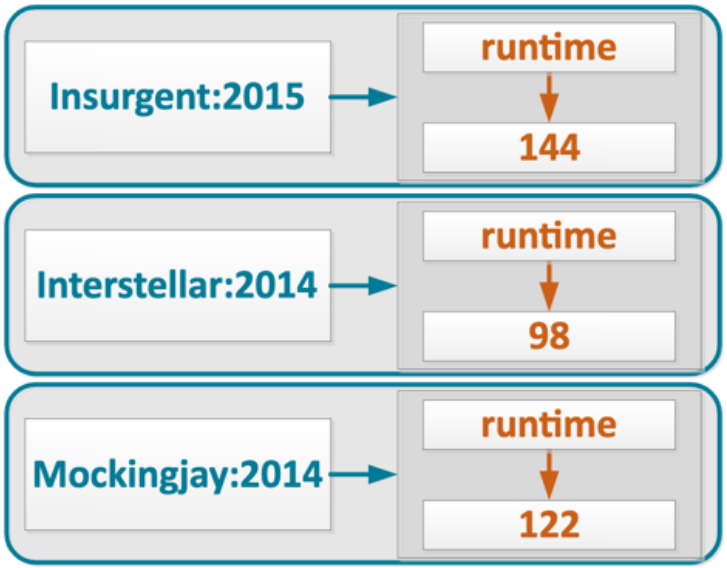
- Contains only the partition key
- Determines which node stores the data



Composite Partition Keys

Multi-value primary key

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



Primary Key vs. Partition Key

Wait! Remind me again what these are?

- **Partition key**: The part of the primary key that determines what node the partition is stored on.
- **Primary key**: Includes partition key and any/all clustering columns.
- Can they be the same? Yes! But not usually.

DATASTAX ACADEMY

Exercise 2.1: Working with Partitions

DATASTAX ACADEMY

Clustering Columns

DATASTAX ACADEMY

Clustering Columns

How Cassandra sorts data within each partition

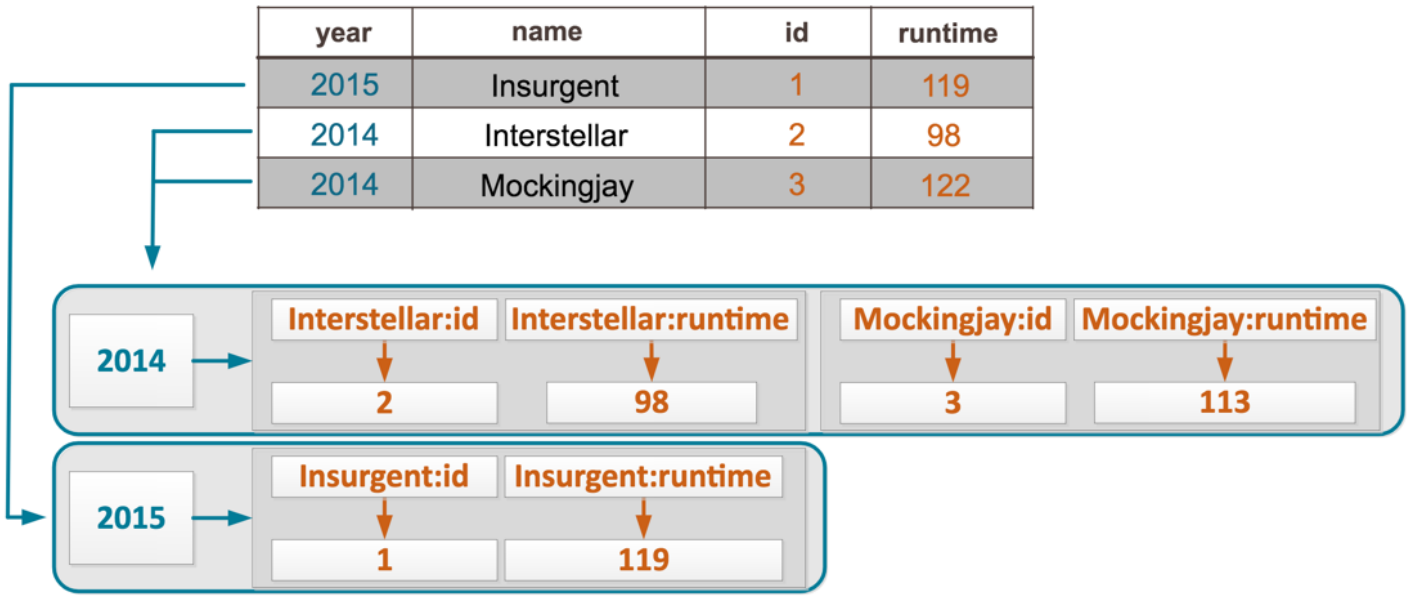
- Come after partition key within PRIMARY KEY clause
- Data displays the same as before

```
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

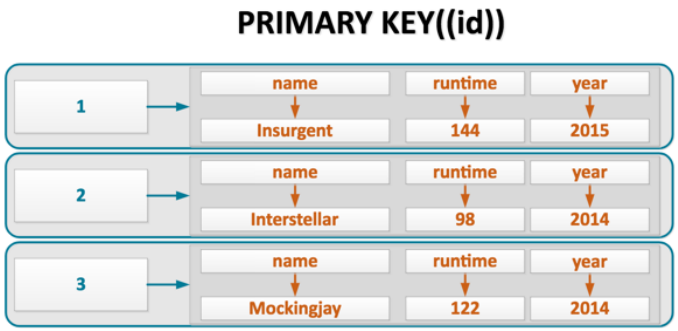
Clustering Columns, Cont.

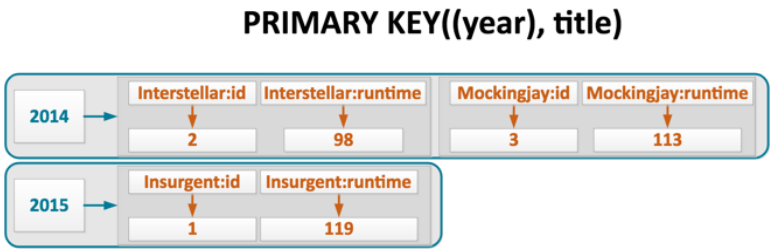
Clustering sorts CQL rows in partitions



Side-by-Side Comparison

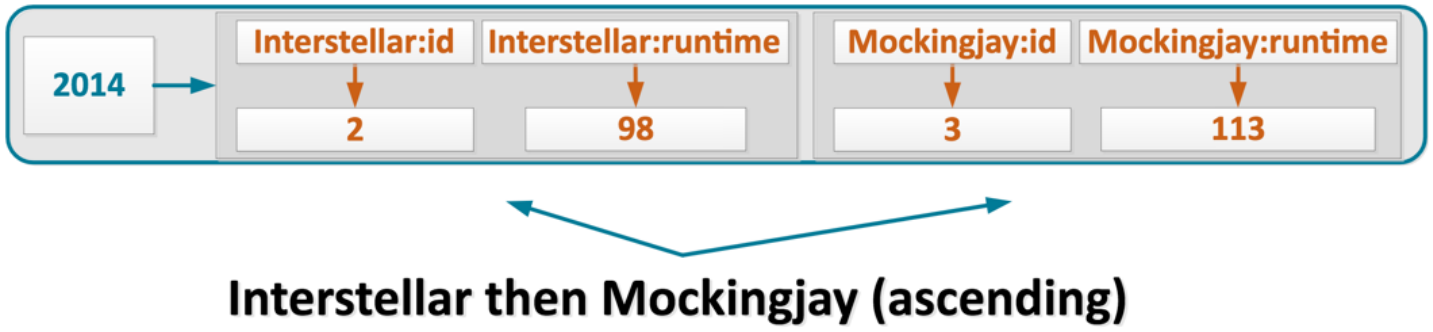
Primary key with just ID vs with year and title.





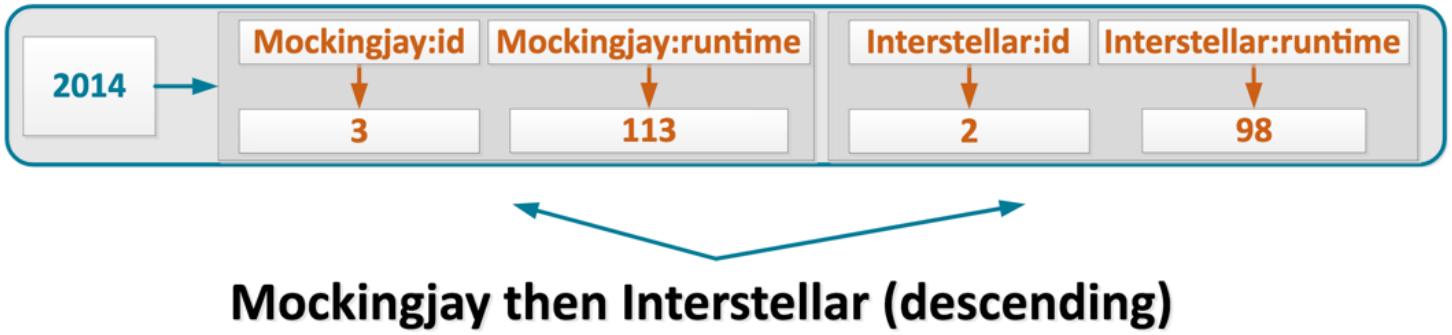
Cluster Column Ordering

Clustering column values stored sorted



Cluster Column Ordering—Descending

Default is ascending but you can specify descending



DATASTAX ACADEMY

Cluster Column Ordering

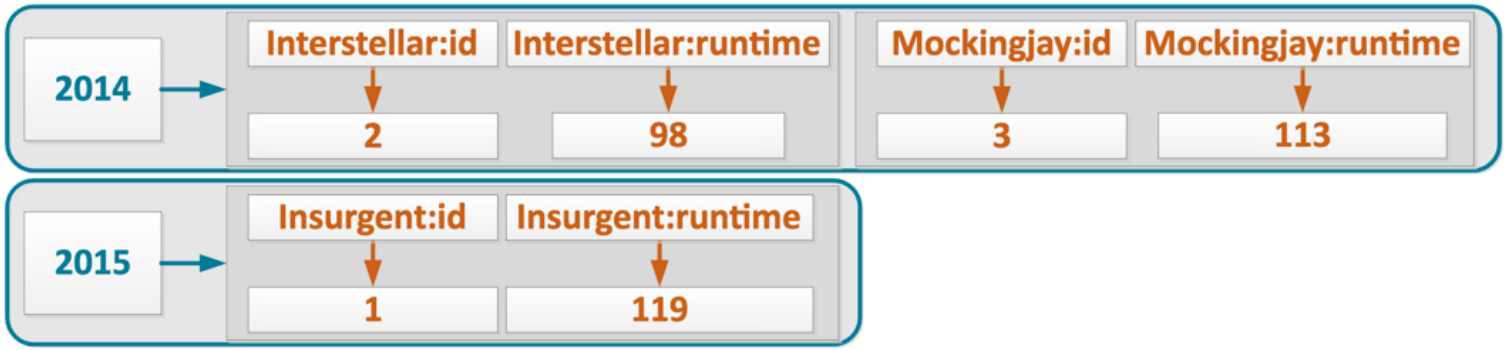
WITH CLUSTERING ORDER BY

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

Querying Clustering Columns

You can query on clustering columns because lookup is fast

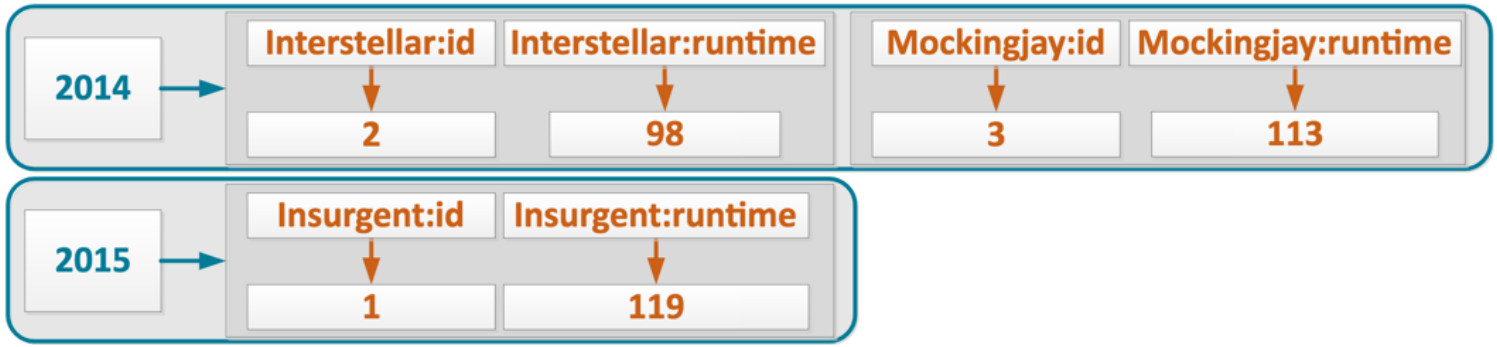
```
SELECT *
FROM videos
WHERE year = 2014 AND name = 'Mockingjay';
```



Querying Clustering Columns—Range

You can also do a range query on clustering columns

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



DATASTAX ACADEMY

Exercise 2.2: Clustering Columns

DATASTAX ACADEMY

Denormalization

Typical Relational Structure

This is a relational example—not Apache Cassandra!

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
…	…	…

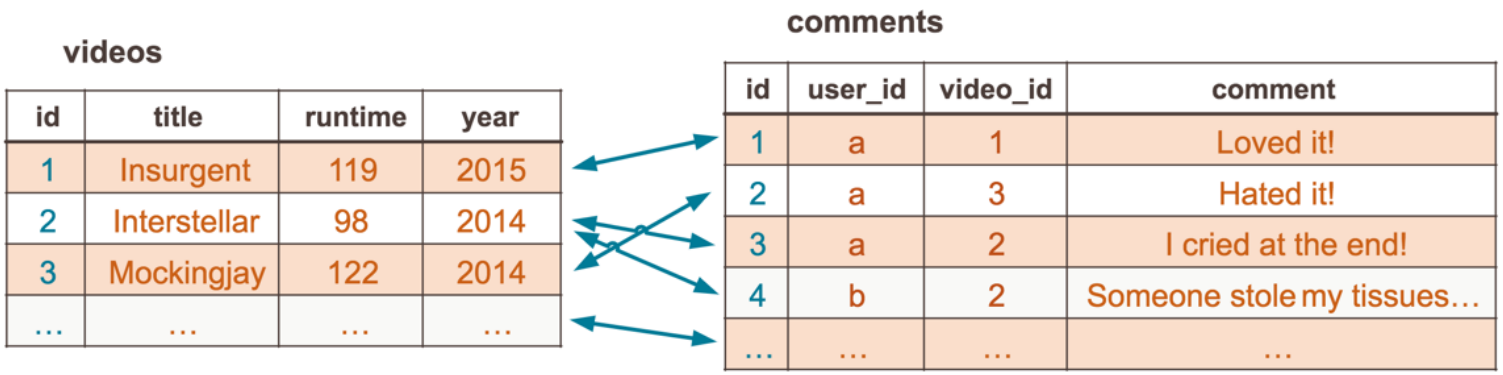
comments

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 Comments By Video Title

Relational Example!

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



videos

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

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 Comments By Video Table

VIDEOS join comments

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

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…
…	…	…	…	…	…	…	…

Query Comments By Video Table

WHERE title = ‘Interstellar’

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

WHERE title = ‘Interstellar’

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…**
…	…	…	…	…	…	…	…

Query Comments By Video Title

Results of the query

```
SELECT comment
FROM videos
JOIN comments
ON videos.id = comments.video_id
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 Comments By User Login

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

users

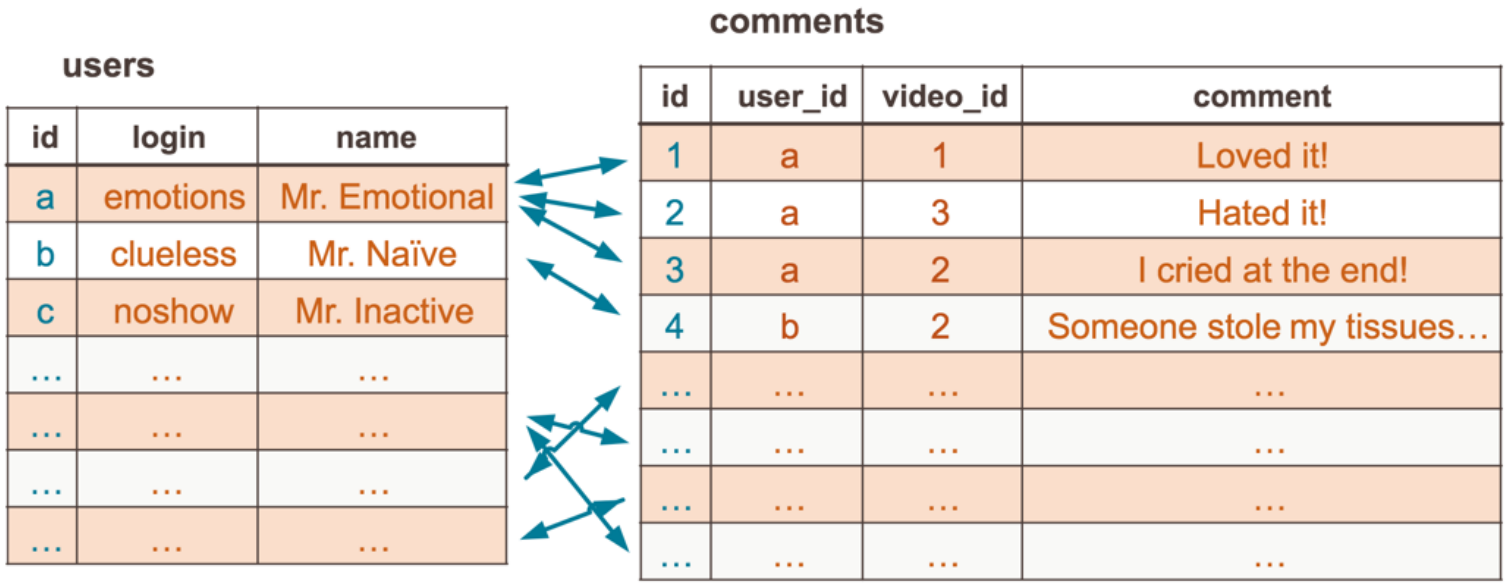
id	login	name
a	emotions	Mr. Emotional
b	clueless	Mr. Naïve
c	noshow	Mr. Inactive

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 Comments By User Login, Cont.

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



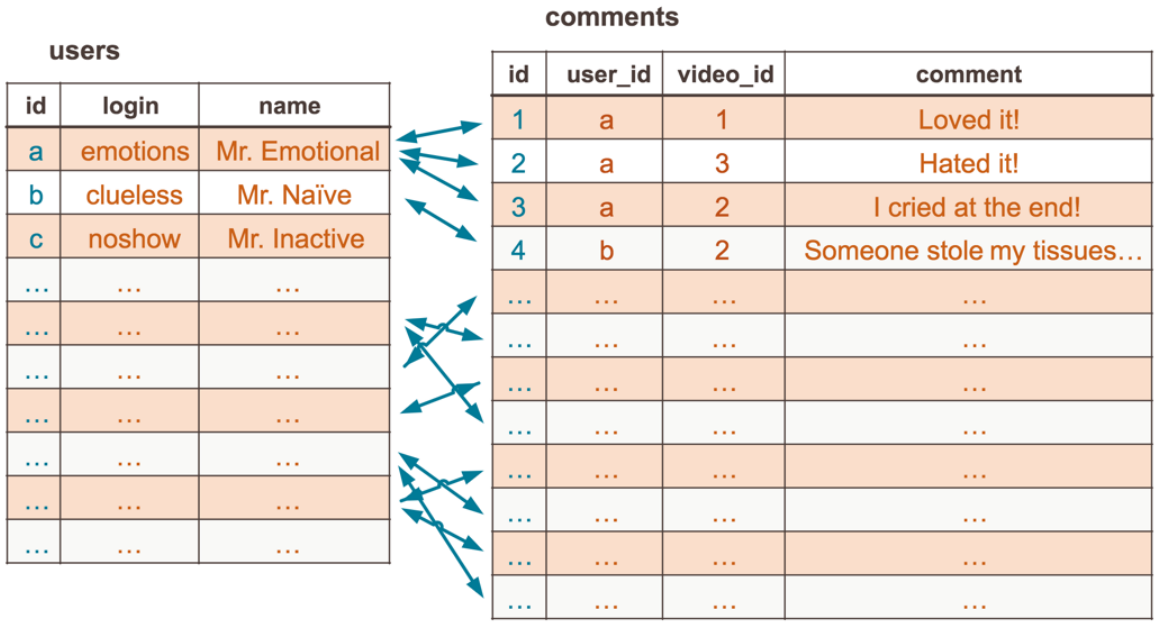
users

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…
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

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



users

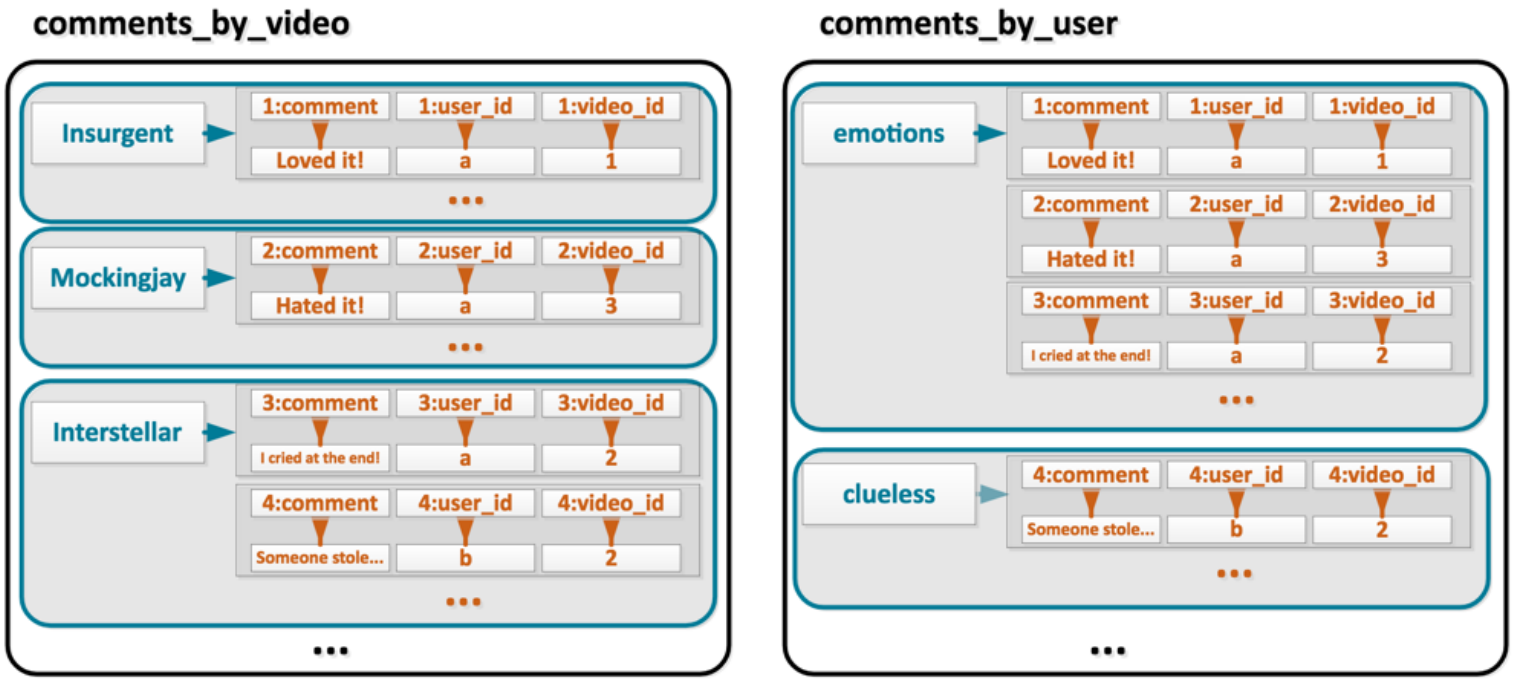
id	login	name
a	emotions	Mr. Emotional
b	clueless	Mr. Naïve
c	noshow	Mr. Inactive
…	…	…
…	…	…
…	…	…
…	…	…
…	…	…
…	…	…
…	…	…

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…
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

Denormalizing For Query Performance

How Apache Cassandra™ works around no joins



Denormalizing For Query Performance

```
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)
);

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)
);
```

Denormalizing For Query Performance

comments_by_video

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

comments_by_user

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

DATASTAX ACADEMY

Exercise 2.3: Denormalizing