Data Modeling with Apache Cassandra™

DuyHai DOAN
Datastax Technical Advocate
Apache Zeppelin™ Committer

1	Data modeling objectives
2	The partition key
3	The clustering column(s)
4	Other critical details
5	Workshop

Data Modeling Objectives

Data modeling objectives

1) Get your data **out of Cassandra**

2) Reduce query latency, make your queries faster

3) Avoid disaster in production

Data modeling objectives

1) Get your data out of Cassandra

2) Reduce query **latency**, make your queries **faster**

3) Avoid disaster in production

Data modeling objectives

1) Get your data out of Cassandra

2) Reduce query latency, make your queries faster

3) Avoid **disaster** in production

Data modeling methodology

Design by query

- first, know your **functional queries** (*find users by xxx*, …)
- then design the **table(s)** for direct access
- **denormalize** if necessary

Output of design phase = **schema.cql**

Then start coding

The partition key

Role

Partition key

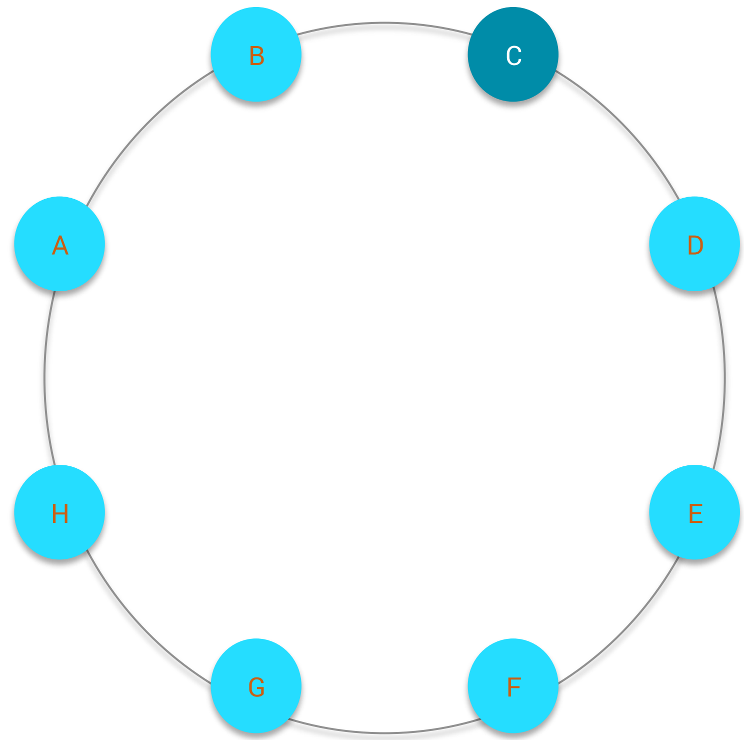
- main entry point for query (INSERT/SELECT …)
- help distribute/locate data on the cluster

No partition key = full cluster scan 😱

Data distribution

Partition key → hash value

$$C: \left]-\frac{2x}{4}, -\frac{x}{4}\right]$$



Hash ranges:

$$A: \left]-x, -\frac{3x}{4}\right] \qquad E: \left]0, \frac{x}{4}\right]$$

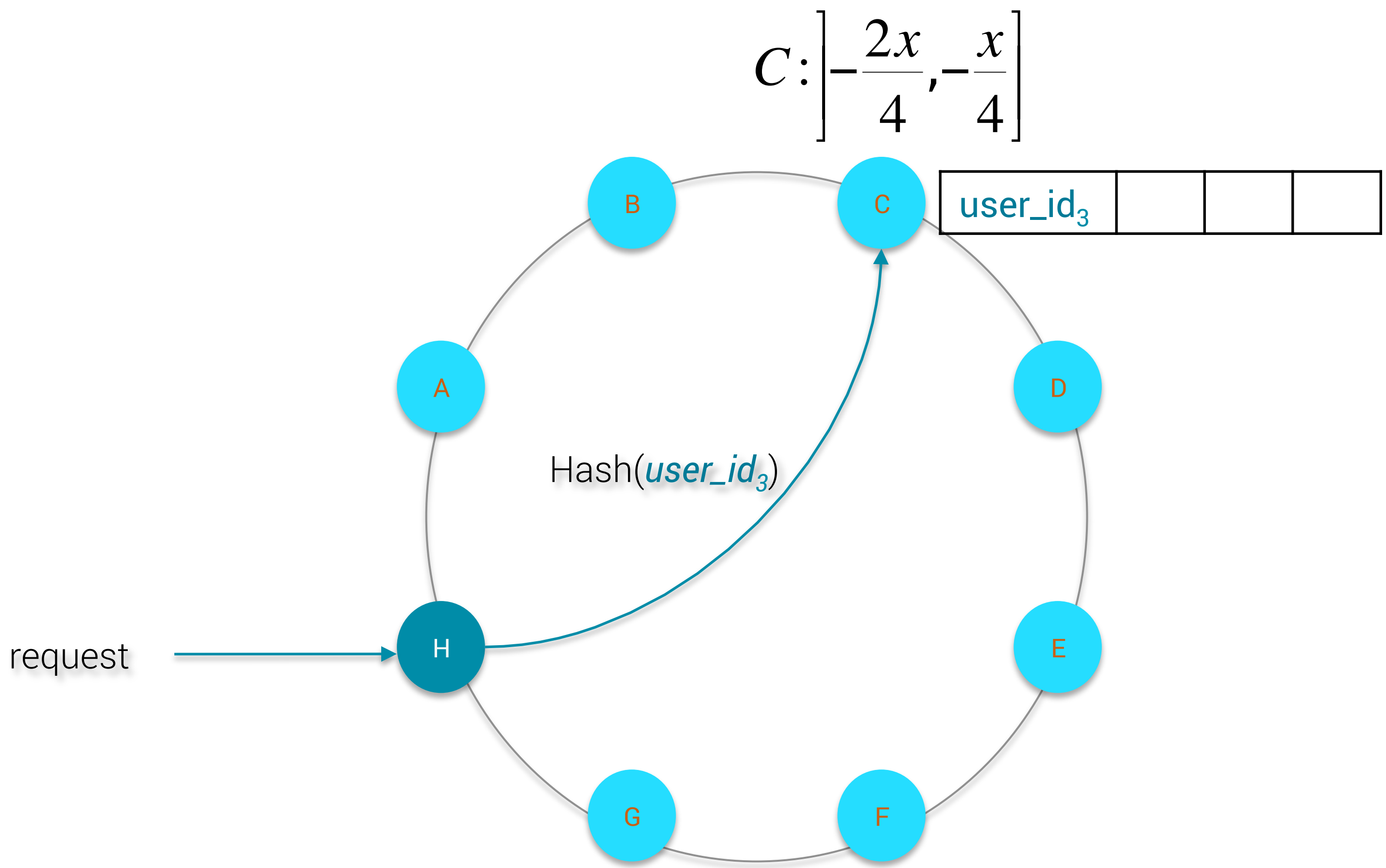
$$B: \left]-\frac{3x}{4}, -\frac{2x}{4}\right] \qquad F: \left]\frac{x}{4}, \frac{2x}{4}\right]$$

$$C: \left]-\frac{2x}{4}, -\frac{x}{4}\right] \qquad G: \left]\frac{2x}{4}, \frac{3x}{4}\right]$$

$$D: \left]-\frac{x}{4}, 0\right] \qquad H: \left]\frac{3x}{4}, x\right]$$

Query by partition key

SELECT * FROM users WHERE user_id = *$user\_id_3$*



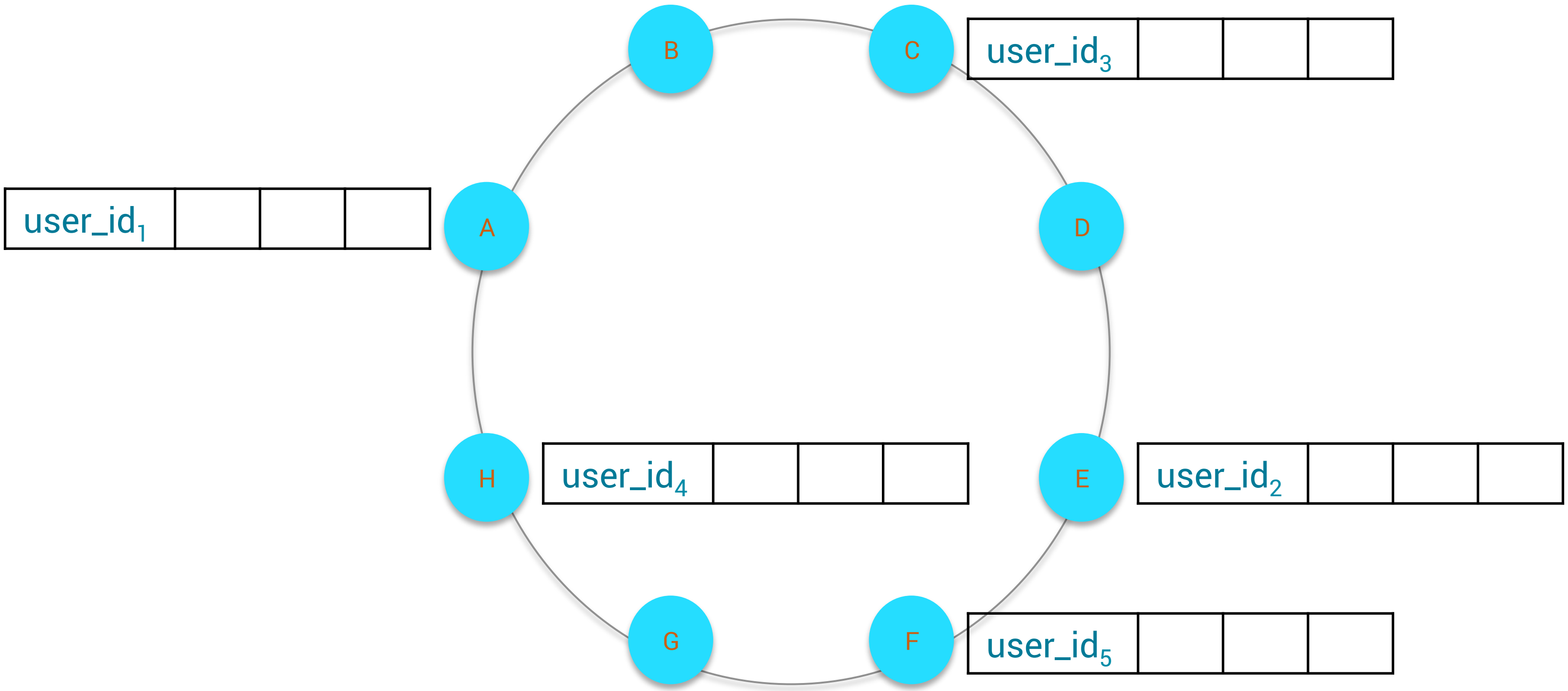
How to choose correct partition key ?

Good partition column

- functional identifier
- high cardinality (lots of distinct values)

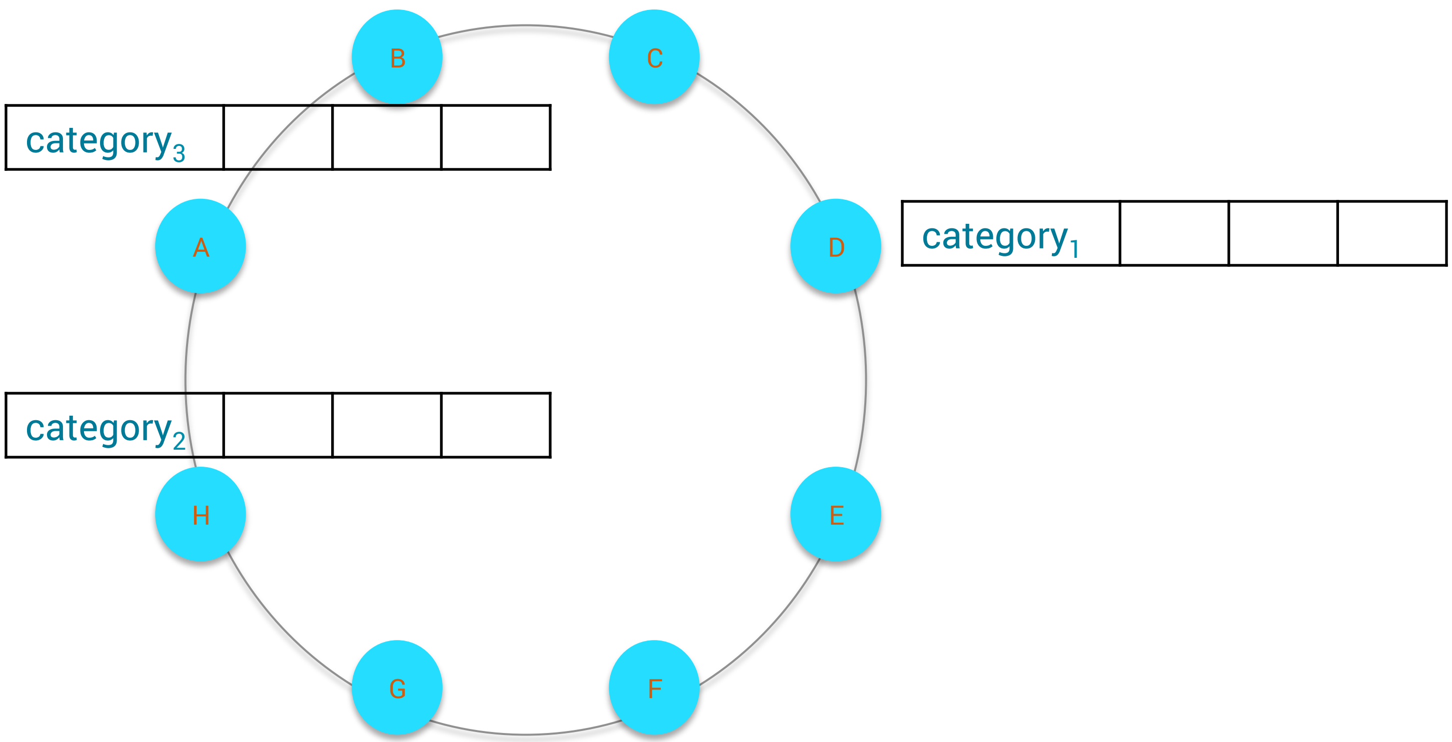
Example of good partition key

```
CREATE TABLE users(
  user_id int,
  ...,
  PRIMARY KEY(user_id))
```



Example of bad partition key

```
CREATE TABLE product(
  category text,
  product_id uuid,
  …,
  PRIMARY KEY((category), product_id));
```



Composite partition key

Multiple columns for partition key

- always **known in advance** (INSERT/SELECT …)
- **hashed together** to the same token value

```
CREATE TABLE product(
  category text,
  product_id uuid,
  product_name text,
  …,
  PRIMARY KEY((category,product_id))
```

```
SELECT * FROM product WHERE category = … AND product_id = …
```

```
SELECT * FROM product WHERE category = … AND product_id IN (xxx, yyy …)
```

The clustering column(s)

Role

Clustering column(s)

- simulate **1 – N relationship**
- **sort** data (logically & on disk)

Example

```
CREATE TABLE sensor_data (
   sensor_id uuid,
   date timestamp,
   value double,
   type text,
   unit text,
PRIMARY KEY(sensor_id, date))
```

Partition key

Clustering column

Recommended syntax

```
PRIMARY KEY((sensor_id), date))
```

Columns relationship

```
CREATE TABLE sensor_data (
   sensor_id uuid,
   date timestamp,
   value double,
   type text,
   unit text,
PRIMARY KEY((sensor_id), date))
```

sensor_id (1) <------------> (N) date

Columns relationship

```
CREATE TABLE sensor_data (
  sensor_id uuid,
  date timestamp,
  value double,
  type text,
  unit text,
PRIMARY KEY((sensor_id), date))
```

date (1) <---------> (1) (value, type, unit)

Clustering column sort

Direct key lookup

```
SELECT * FROM sensor_data
WHERE sensor_id = … AND date = '2016-06-14 10:00:00.000'
```

Range queries

```
SELECT * FROM sensor_data
WHERE sensor_id = … AND date >= '2016-06-14 10:00:00.000'
```

```
SELECT * FROM sensor_data
WHERE sensor_id = … AND date <= '2016-06-14 10:00:00.000'
```

```
SELECT * FROM sensor_data
WHERE sensor_id = …
AND date >= '2016-06-14 10:00:00.000'
AND date <= '2016-06-14 12:00:00.000'
```

On-disk clustering column sort

sensor_id$_1$	2016-06-14 10:00:00.0000			2016-06-14 11:00:00.0000			...		
	23.4	Temperature	Celsius	23.7	Temperature	Celsius	...	...	...

sensor_id$_7$	2016-06-14 10:00:00.0000			2016-06-14 11:00:00.0000			...		
	21.3	Temperature	Celsius	20.1	Temperature	Celsius	...	...	...

File 1

sensor_id$_5$	2016-06-14 10:00:00.0000			2016-06-14 11:00:00.0000			...		
	23.4	Temperature	Celsius	23.7	Temperature	Celsius	...	...	...

sensor_id$_2$	2016-06-14 10:00:00.0000			2016-06-14 11:00:00.0000			...		
	21.3	Temperature	Celsius	20.1	Temperature	Celsius	...	...	...

File 2

Reverse clustering sort

```
CREATE TABLE sensor_data (
   sensor_id uuid,
   date timestamp,
   value double,
   type text,
   unit text,
PRIMARY KEY((sensor_id), date))
WITH CLUSTERING ORDER BY (date DESC)
```

$sensor\_id_1$	2016-06-14 11:00:00.0000			2016-06-14 10:00:00.0000			...		
	23.7	Temperature	Celsius	23.4	Temperature	Celsius	...	...	...

Multiple clustering columns

```
CREATE TABLE sensor_data (
   sensor_id uuid,
   priority int,
   date timestamp,
   value double,
   type text,
   unit text,
PRIMARY KEY((sensor_id), priority, date))
WITH CLUSTERING ORDER BY (priority ASC, date DESC)
```

priority (1) <---------> (N) date

<table>
<tr><td rowspan="3">sensor_id₁</td><td colspan="6">1</td><td colspan="3">2</td></tr>
<tr><td colspan="3">2016-06-14 <u>11</u>:00:00.0000</td><td colspan="3">2016-06-14 <u>10</u>:00:00.0000</td><td colspan="3">...</td></tr>
<tr><td>23.7</td><td>Temperature</td><td>Celsius</td><td>23.4</td><td>Temperature</td><td>Celsius</td><td>...</td><td>...</td><td>...</td></tr>
</table>

Multiple clustering columns

```
PRIMARY KEY((sensor_id), priority, date))
```

Nested map abstraction:

```
Map<sensor_id,
    SortedMap<priority,
        SortedMap<date, (value,type,unit)>>>
```

```
WHERE sensor_id = … AND priority = … AND date = …
```

```
WHERE sensor_id = … AND priority = … AND date >= … AND date <= …
```

```
WHERE sensor_id = … AND priority >= … AND priority <=
```

Primary key summary

```
PRIMARY KEY((sensor_id), priority, date))
```

$\underbrace{\text{sensor\_id), priority, date}}_{}$

Unicity of (sensor_id, priority, date)

DATASTAX

Primary key summary

$$\text{PRIMARY KEY}((\underbrace{\text{sensor\_id}}_{\text{Used to locate node in the cluster}}), \text{priority}, \text{date}))$$

Used to locate node in the cluster
Used to locate partition in the node

Primary key summary

PRIMARY KEY((sensor_id), $\underbrace{\text{priority, date}}$))

Used to lookup rows in a partition
Used for data sorting and range queries

Other critical details

Huge partitions

```
PRIMARY KEY((sensor_id), date))
```

Data for the **same sensor** stay in the **same partition** on disk

Huge partitions

PRIMARY KEY((sensor_id), date))

Data for the **same sensor** stay in the **same partition** on disk

If insert rate = 100/sec, how big is my partition after 1 year ?

→ 100 x 3600 x 24 x 365= **3 153 600 000** cells on disks

Huge partitions

PRIMARY KEY((sensor_id), date))

Theorical limit of # cells for a partition = 2×10^9

Practical limit for a partition on disk

- 100Mb
- 100 000 – 1000 000 cells

Reasons ? Make maintenance operations easier

- compaction
- repair
- bootstrap …

Sub-partitioning techniques

```
PRIMARY KEY((sensor_id, day), date))
```

→ 100 x 3600 x 24 = **8 640 000** cells on disks ✔

Sub-partitioning techniques

```
PRIMARY KEY((sensor_id, day), date))
```

→ 100 x 3600 x 24 = **8 640 000** cells on disks ✔

But impact on queries:

- need to provide **sensor_id** & **day** for any query
- how to fetch data across N days ?

Data deletion and tombstones

```
DELETE FROM sensor_data
WHERE sensor_id = .. AND date = …
```

Logical deletion of data but:

- new physical "tombstone" column on disk
- **disk space usage will increase !**

The "tombstone" columns will be purged later by compaction process …

Workshop

Workshop setup

Full setup and materials are here

https://github.com/julienmichel/CapGeminiWorkshop