Introduction to Data Modeling

Patrick McFadin

Chief Evangelist for Apache Cassandra

@PatrickMcFadin

My Background

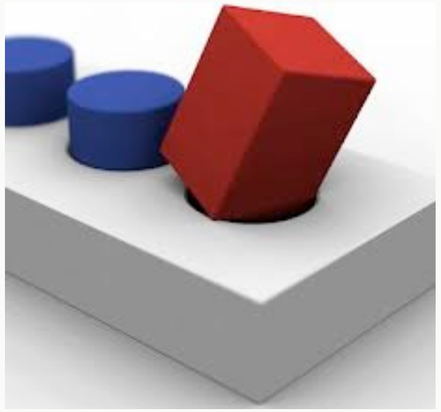
Microsoft® SQL Server®

MySQL®

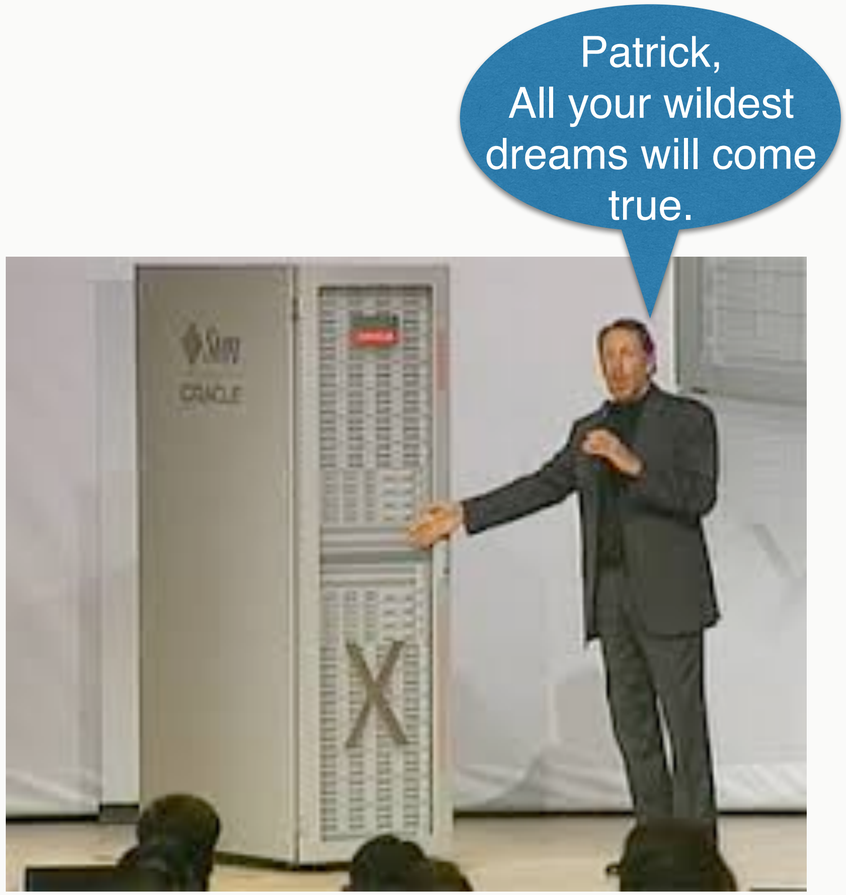
IBM DB2

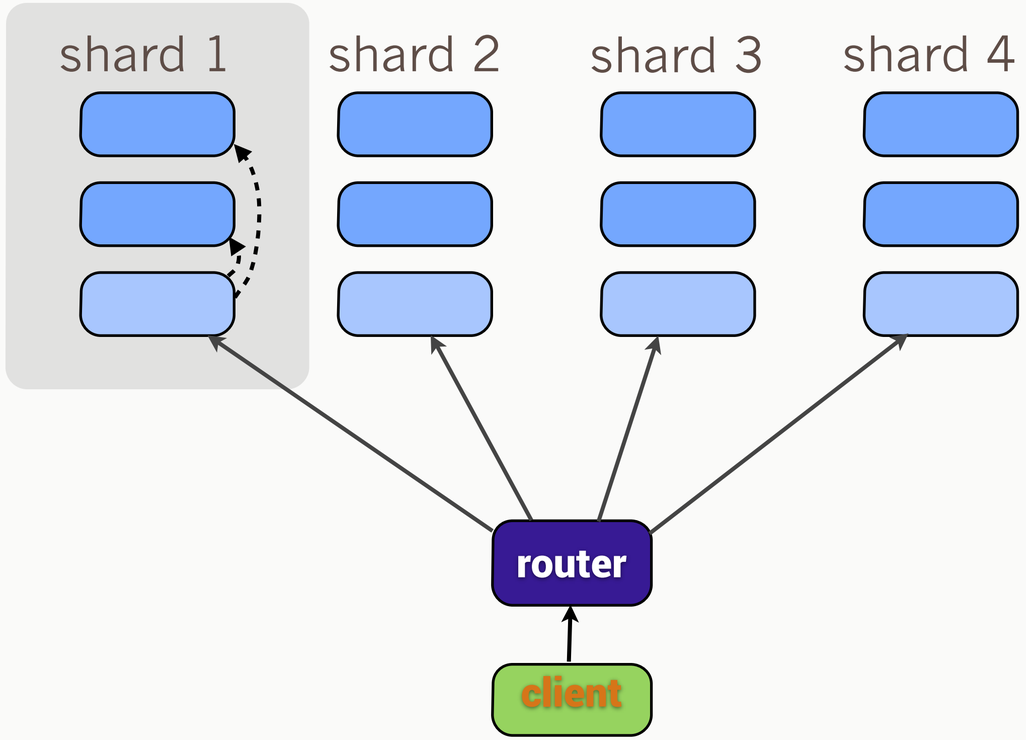
PostgreSQL

…ran into this problem



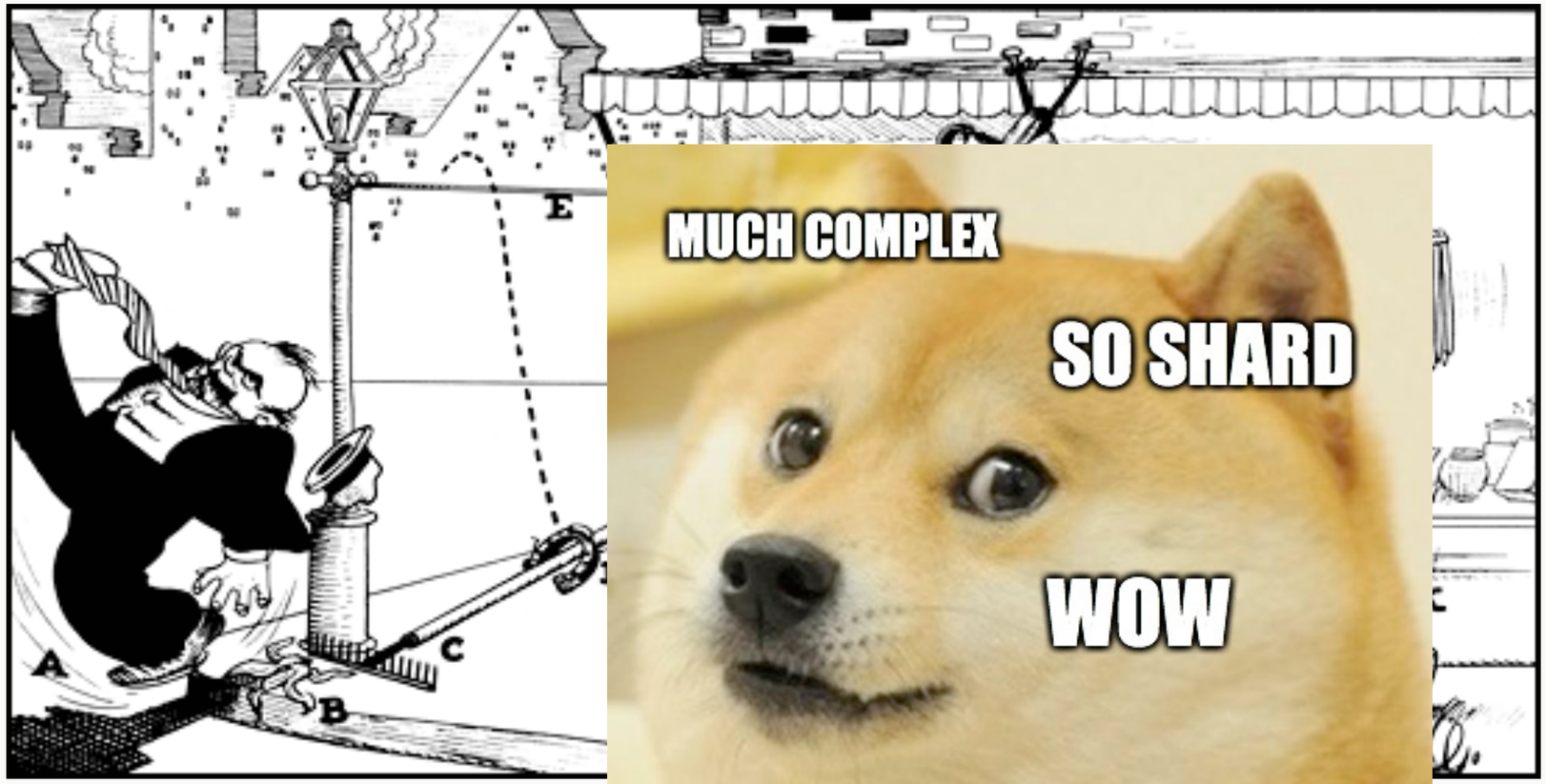
Gave it my best shot





Just add complexity!

DATASTAX



A new plan

ACID vs CAP

DATASTAX

ACID

Atomic - All or none

Consistency - Only valid data is written

Isolation - One operation at a time

Durability - Once committed, it stays that way

CAP - Pick two

Cassandra let's you tune this

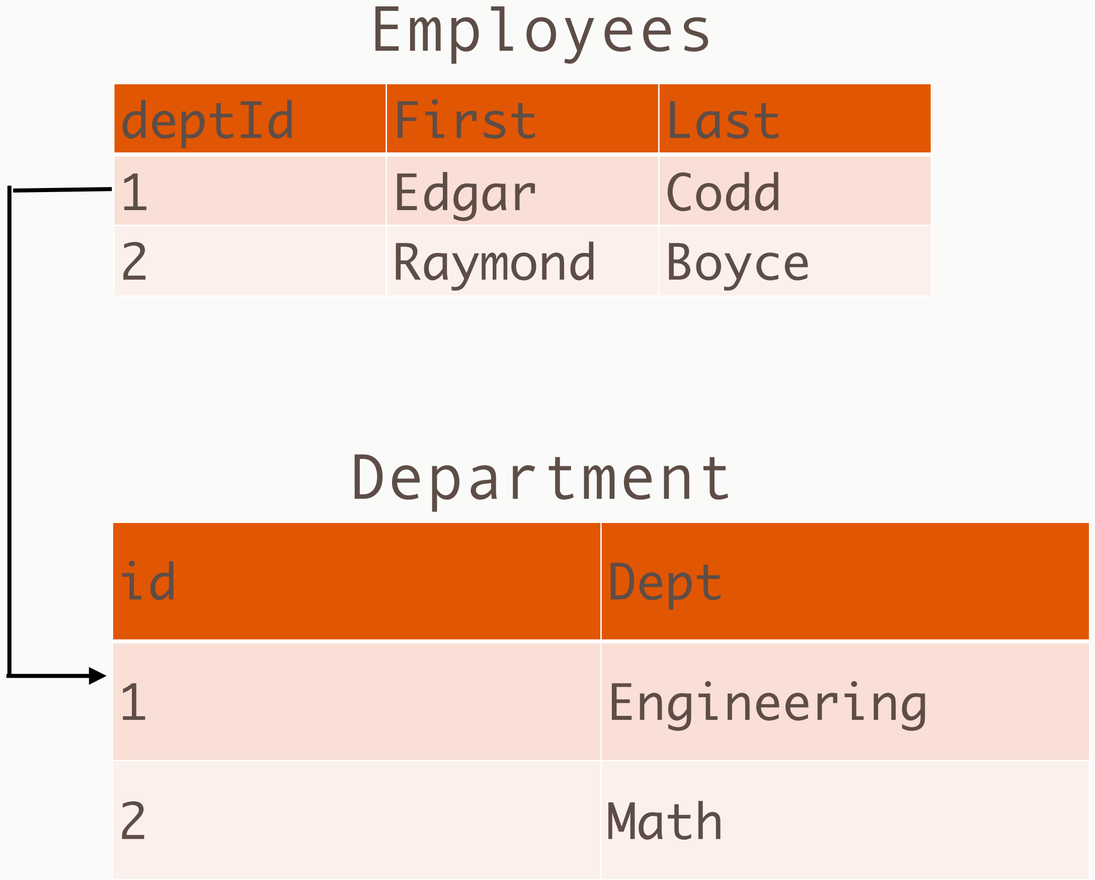
Consistency - All data on cluster

Availability - Cluster always accepts writes

Partition tolerance - Nodes in cluster can't talk to each other

Relational Data Models

- 5 normal forms
- Foreign Keys
- Joins



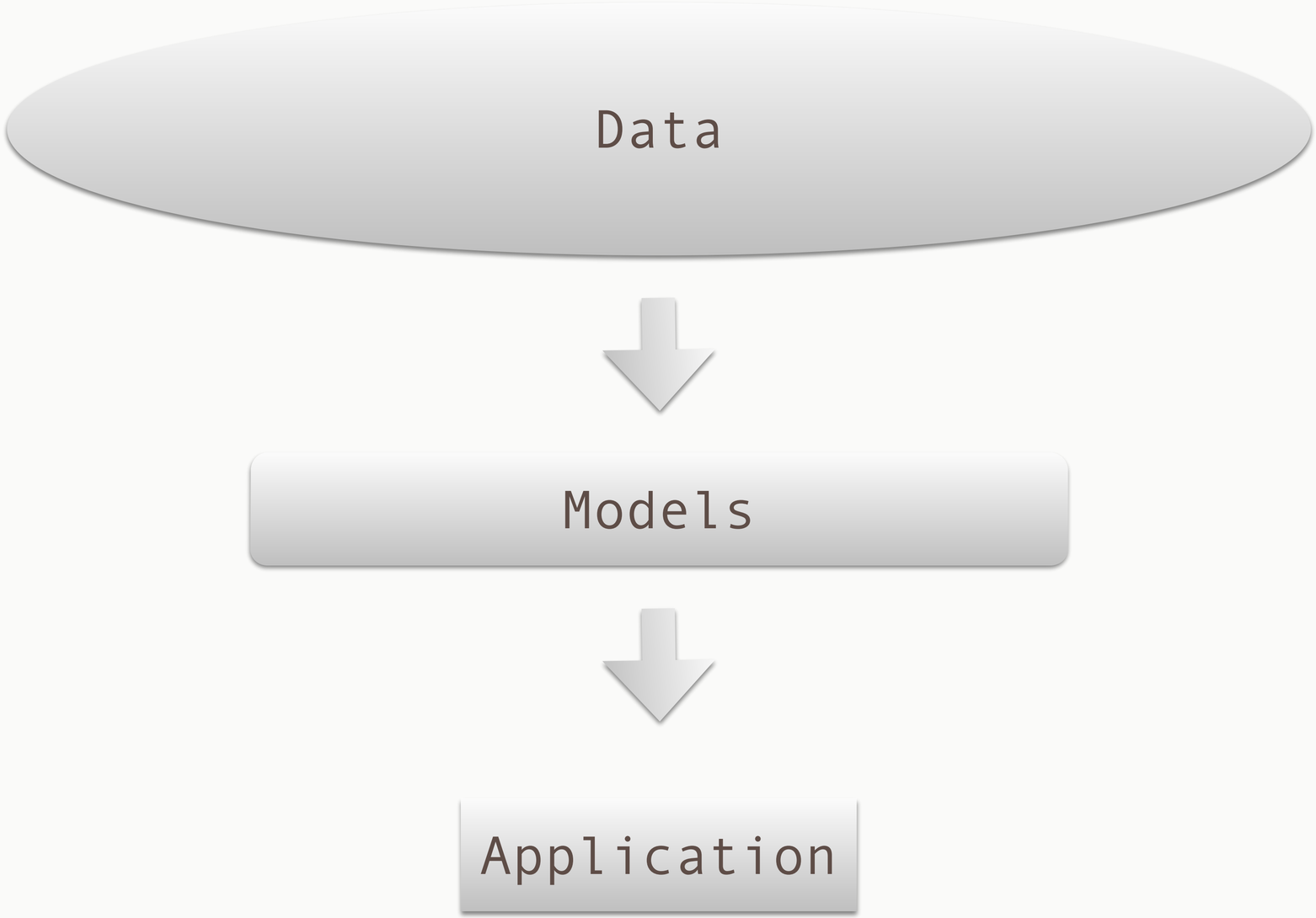
Employees

deptId	First	Last
1	Edgar	Codd
2	Raymond	Boyce

Department

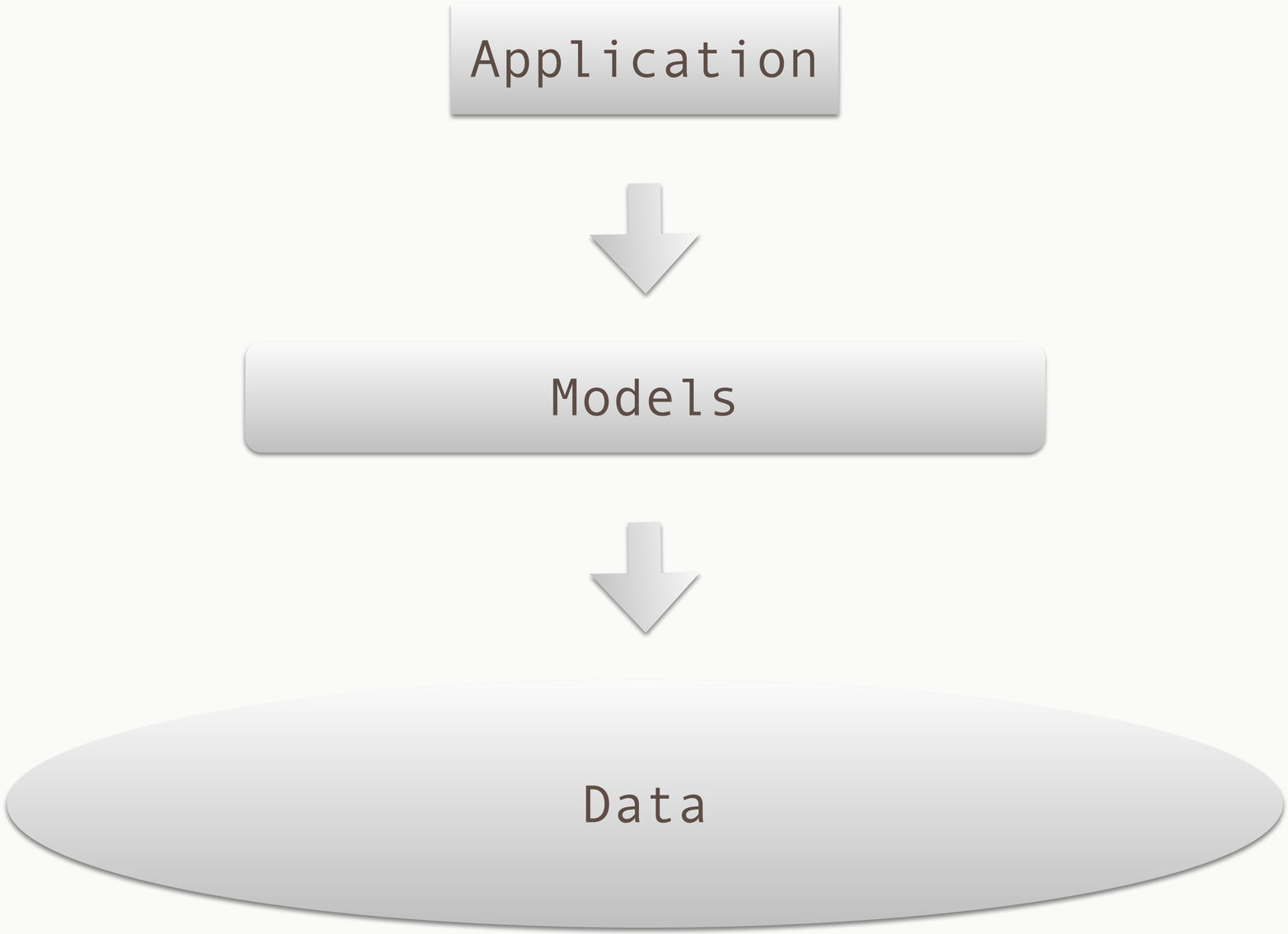
id	Dept
1	Engineering
2	Math

Relational Modeling



Cassandra Modeling

DATASTAX



CQL vs SQL

- No joins
- No aggregations

```
SELECT e.First, e.Last, d.Dept
FROM Department d, Employees e
WHERE ‘Codd’ = e.Last
AND e.deptId = d.id
```

Employees

deptId	First	Last
1	Edgar	Codd
2	Raymond	Boyce

Department

id	Dept
1	Engineering
2	Math

Denormalization

- Combine table columns into a single view
- No joins

Employees

id	First	Last	Dept
1	Edgar	Codd	Engineering
2	Raymond	Boyce	Math

```
SELECT First, Last, Dept
FROM employees
WHERE id = ‘1’
```

No more sequences

DATASTAX

- Great for auto-creation of Ids
- Guaranteed unique
- Needs ACID to work. (Sorry. No sharding)

```
INSERT INTO user (id, firstName, LastName)
VALUES (seq.nextVal(), ‘Ted’, ‘Codd’)
```

No sequences???

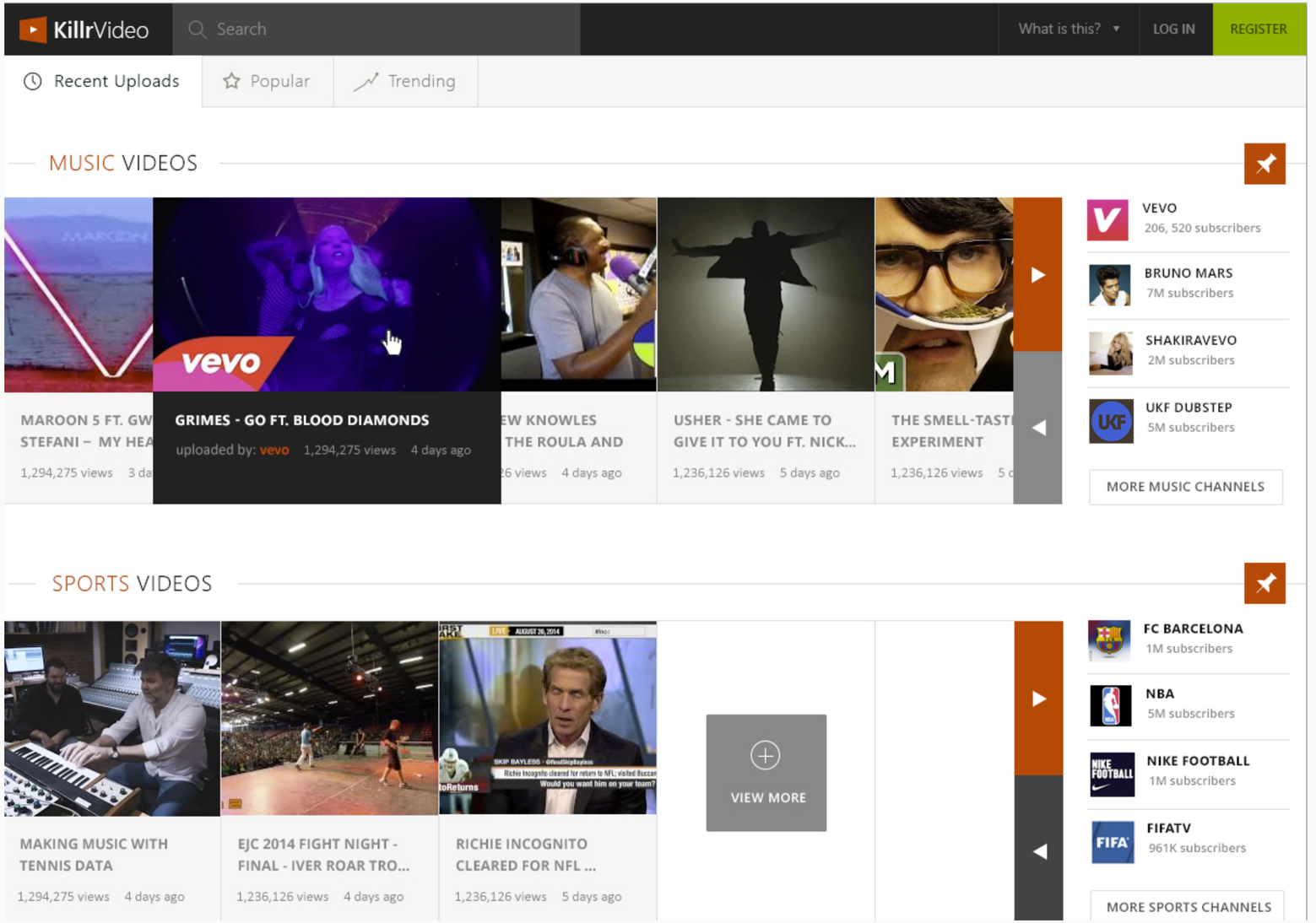
DATASTAX

- Almost impossible in a distributed system
- Couple of great choices
  - Natural Key - Unique values like email
  - Surrogate Key - UUID
    - Universal Unique ID
    - 128 bit number represented in character form
    - Easily generated on the client
    - Same as GUID for the MS folks

```
99051fe9-6a9c-46c2-b949-38ef78858dd0
```

KillrVideo.com

- Hosted on Azure
- Code on GitHub
- Also on your USB
- Data Model for examples



Entity Table

DATASTAX

- Simple view of a single user
- UUID used for ID
- Simple primary key

```
// Users keyed by id
CREATE TABLE users (
    userid uuid,
    firstname text,
    lastname text,
    email text,
    created_date timestamp,
    PRIMARY KEY (userid)
);

SELECT firstname, lastname
FROM user
WHERE userId = 99051fe9-6a9c-46c2-b949-38ef78858dd0
```

CQL Collections

CQL Collections

DATASTAX

- Meant to be dynamic part of table
- Update syntax is very different from insert
- Reads require all of collection to be read

CQL Set

- Set is sorted by CQL type comparator


```
INSERT INTO collections_example (id, set_example)
VALUES(1, {'1-one', '2-two'});
```

CQL Set Operations

DATASTAX

- Adding an element to the set

```
UPDATE collections_example
SET set_example = set_example + {'3-three'} WHERE id = 1;
```

- After adding this element, it will sort to the beginning.

```
UPDATE collections_example
SET set_example = set_example + {'0-zero'} WHERE id = 1;
```

- Removing an element from the set

```
UPDATE collections_example
SET set_example = set_example - {'3-three'} WHERE id = 1;
```

CQL List

- Ordered by insertion
- Use with caution

```
list_example list<text>
```

Collection name Collection type CQL Type

```
INSERT INTO collections_example (id, list_example)
VALUES(1, ['1-one', '2-two']);
```

CQL List Operations

DATASTAX

- Adding an element to the end of a list

```
UPDATE collections_example
SET list_example = list_example + ['3-three']
WHERE id = 1;
```

- Adding an element to the beginning of a list

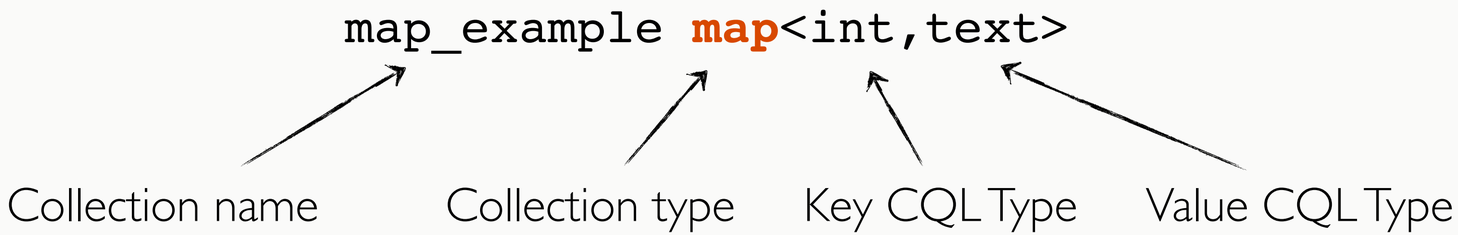
```
UPDATE collections_example
SET list_example = ['0-zero'] + list_example
WHERE id = 1;
```

- Deleting an element from a list

```
UPDATE collections_example
SET list_example = list_example - ['3-three'] WHERE id = 1;
```

CQL Map

- Key and value
- Key is sorted by CQL type comparator




```
INSERT INTO collections_example (id, map_example)
VALUES(1, { 1 : 'one', 2 : 'two' });
```

CQL Map Operations

DATASTAX

- Add an element to the map

```
UPDATE collections_example
SET map_example[3] = 'three'
WHERE id = 1;
```

- Update an existing element in the map

```
UPDATE collections_example
SET map_example[3] = 'tres'
WHERE id = 1;
```

- Delete an element in the map

```
DELETE map_example[3]
FROM collections_example
WHERE id = 1;
```

Entity with collections

DATASTAX

- Same type of entity
- SET type for dynamic data
- tags for each video

```
// Videos by id
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)
);
```

Index (or lookup) tables

DATASTAX

- Table arranged to find data
- Denormalized for speed
- Find videos for a user

```
// One-to-many from user point of view (lookup table)
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);
```

Primary Key

DATASTAX

- First column name is the Partition Key
- Subsequent are the Clustering Columns
- Videos will be ordered by added_date and

```
// One-to-many from user point of view (lookup table)
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);
```

Primary key relationship

DATASTAX

```
PRIMARY KEY (userId,added_date,videoId)
```

Primary key relationship

DATASTAX

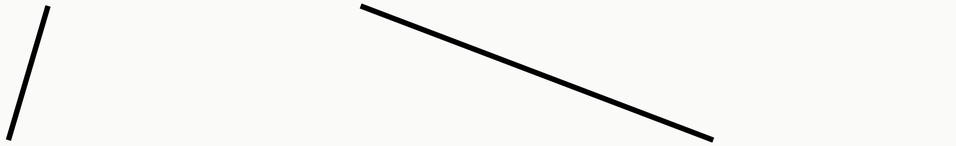
```
PRIMARY KEY (userId,added_date,videoId)
```

Partition Key

Primary key relationship

DATASTAX

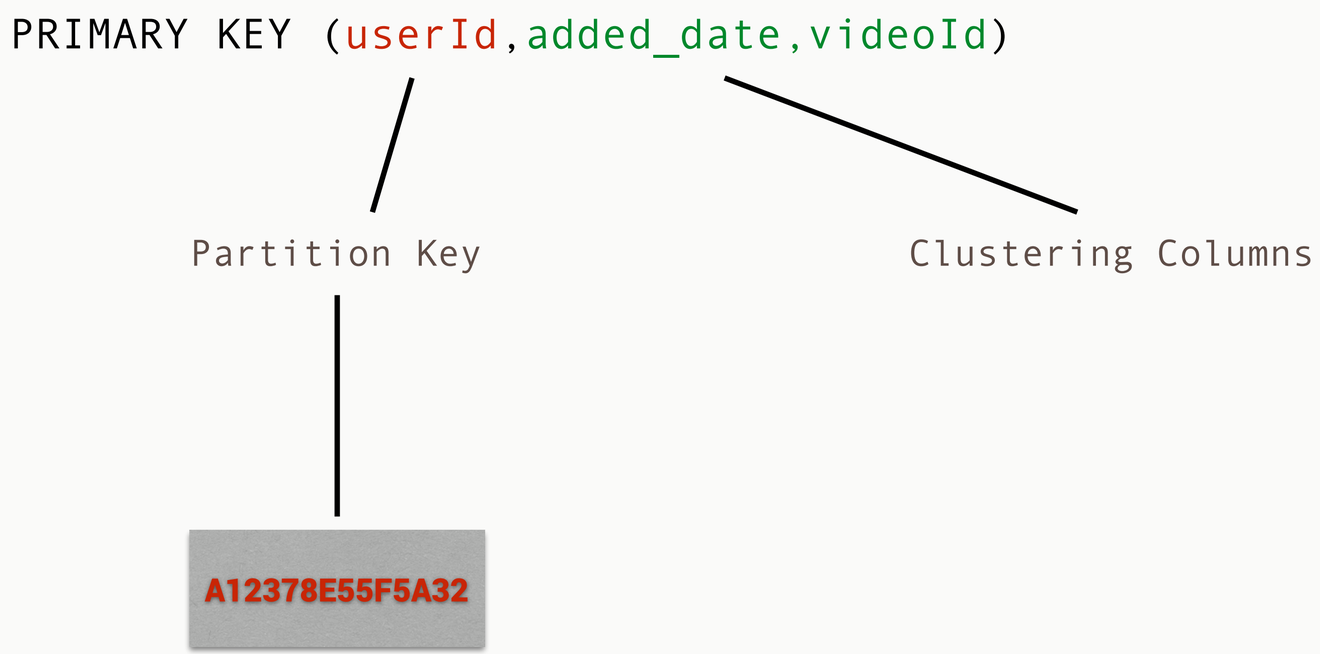
```
PRIMARY KEY (userId,added_date,videoId)
```



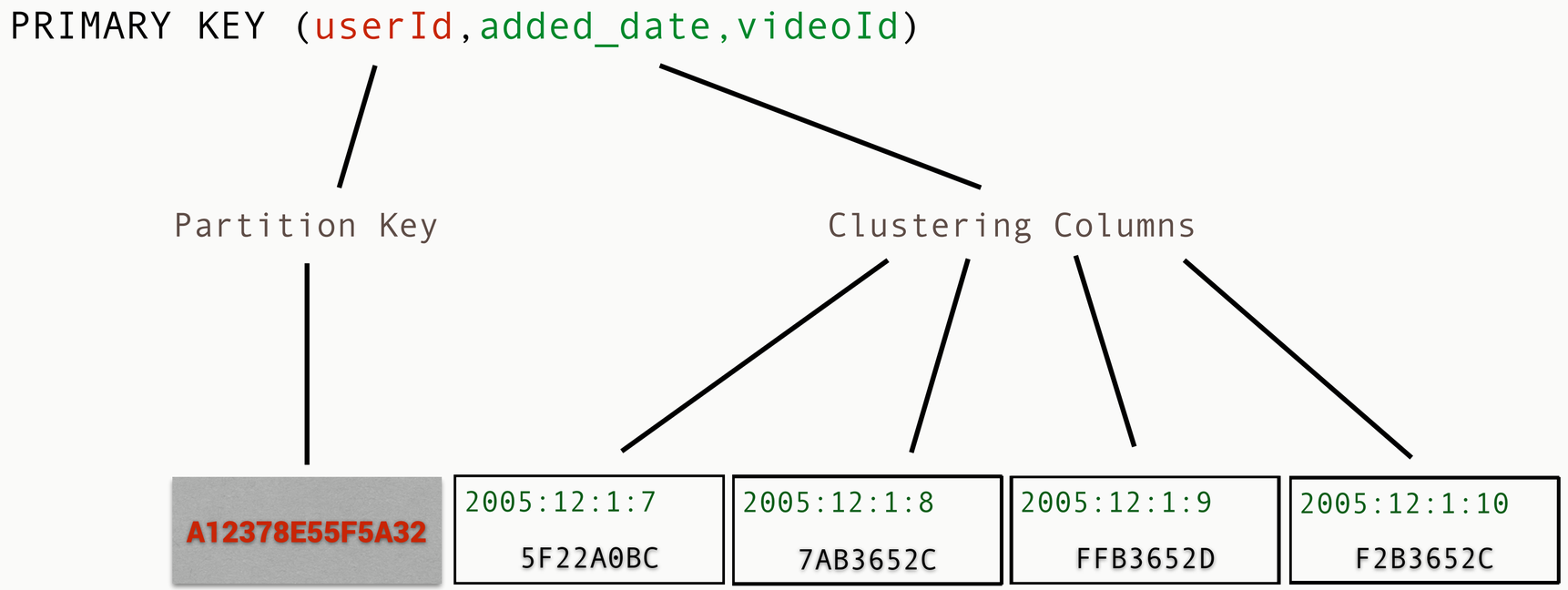
Partition Key

Clustering Columns

Primary key relationship



Primary key relationship




```
SELECT videoId FROM user_videos
WHERE userId = A12378E55F5A32

AND added_date = ‘2005-12-1’

AND videoId = 5F22A0BC
```

Clustering Order

DATASTAX

- Clustering Columns have default order
- Use to specify order
- Bonus: Sorts on disk for speed

```
// One-to-many from user point of view (lookup table)
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);
```

Multiple Lookups

DATASTAX

- Same data
- Different lookup pattern

```
// Index for tag keywords
CREATE TABLE videos_by_tag (
    tag text,
    videoid uuid,
    added_date timestamp,
    name text,
    preview_image_location text,
    tagged_date timestamp,
    PRIMARY KEY (tag, videoid)
);

// Index for tags by first letter in the tag
CREATE TABLE tags_by_letter (
    first_letter text,
    tag text,
    PRIMARY KEY (first_letter, tag)
);
```

Many to Many Relationships

DATASTAX

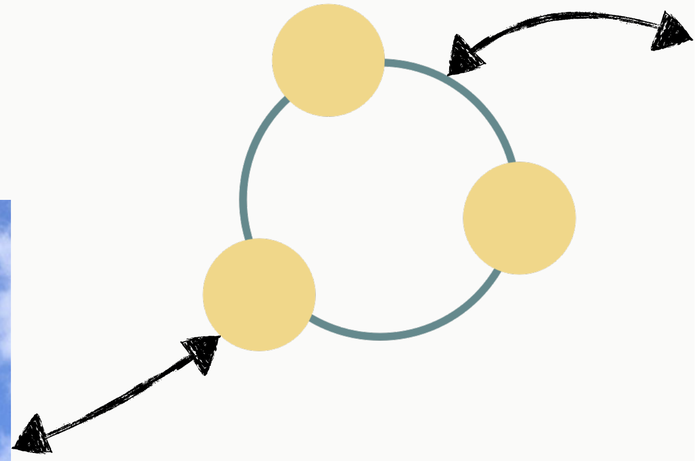
- Two views
- Different directions
- Insert data in a batch

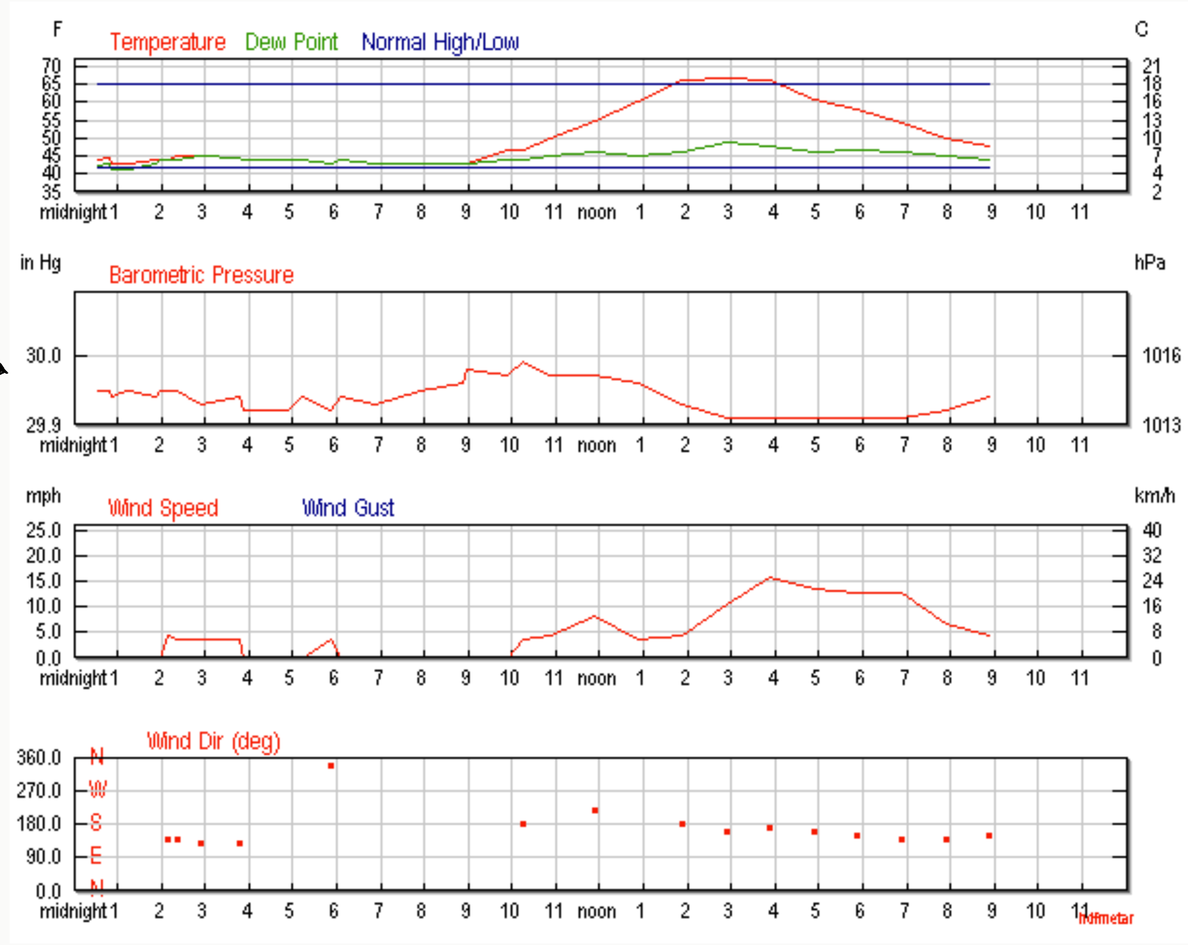
```
// Comments for a given video
CREATE TABLE comments_by_video (
    videoid uuid,
    commentid timeuuid,
    userid uuid,
    comment text,
    PRIMARY KEY (videoid, commentid)
) WITH CLUSTERING ORDER BY (commentid DESC);

// Comments for a given user
CREATE TABLE comments_by_user (
    userid uuid,
    commentid timeuuid,
    videoid uuid,
    comment text,
    PRIMARY KEY (userid, commentid)
) WITH CLUSTERING ORDER BY (commentid DESC);
```

Use Case Example

Example 1: Weather Station

- Weather station collects data
- Cassandra stores in sequence
- Application reads in sequence





Use case

DATASTAX

Needed Queries

- Get all data for one weather station
- Get data for a single date and time
- Get data for a range of dates and times

Data Model to support queries

- Store data per weather station
- Store time series in order: first to last

Data Model

DATASTAX

```
CREATE TABLE temperature (
    weather_station text,
    year int,
    month int,
    day int,
    hour int,
    temperature double,
    PRIMARY KEY (weather_station,year,month,day,hour)
);
```

- Weather Station Id and Time are unique
- Store as many as needed

```
INSERT INTO temperature(weather_station,year,month,day,hour,temperature)
VALUES ('10010:99999',2005,12,1,7,-5.6);

INSERT INTO temperature(weather_station,year,month,day,hour,temperature)
VALUES ('10010:99999',2005,12,1,8,-5.1);

INSERT INTO temperature(weather_station,year,month,day,hour,temperature)
VALUES ('10010:99999',2005,12,1,9,-4.9);

INSERT INTO temperature(weather_station,year,month,day,hour,temperature)
VALUES ('10010:99999',2005,12,1,10,-5.3);
```

Storage Model - Logical View

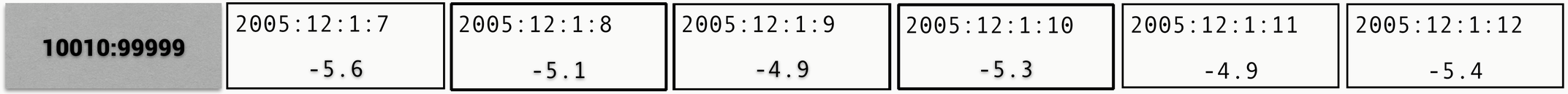
DATASTAX

```
SELECT weather_station,hour,temperature
FROM temperature
WHERE weatherstation_id='10010:99999';
```

weather_station	hour	temperature
10010:99999	2005:12:1 7	-5.6
10010:99999	2005:12:1 8	-5.1
10010:99999	2005:12:1 9	-4.9
10010:99999	2005:12:1 10	-5.3

Storage Model - Disk Layout

```
SELECT weather_station,hour,temperature
FROM temperature
WHERE weatherstation_id='10010:99999';
```



Merged, Sorted and Stored Sequentially

Query patterns

```
SELECT weatherstation,hour,temperature
FROM temperature
WHERE weatherstation='10010:99999'
AND year = 2005 AND month = 12 AND day = 1
AND hour >= 7 AND hour <= 10;
```

- Range queries
- “Slice” operation on disk

Query patterns

```
SELECT weatherstation,hour,temperature
FROM temperature
WHERE weatherstation='10010:99999'
AND year = 2005 AND month = 12 AND day = 1
AND hour >= 7 AND hour <= 10;
```

- Range queries
- “Slice” operation on disk

weather_station	hour	temperature
10010:99999	2005:12:1 7	-5.6
10010:99999	2005:12:1 8	-5.1
10010:99999	2005:12:1 9	-4.9
10010:99999	2005:12:1 10	-5.3

Sorted by event_time

Programmers like this

Thank you!

Bring the questions

Follow me on twitter
@PatrickMcFadin