O'REILLY®
Velocity

DATA MODELING *in the* 24 ½ CENTURY

Amanda Moran
Developer Advocate DataStax

velocityconf.com/ca
#VelocityConf

The Future Is Here!

Actually it's been here…

NoSQL has been around since the early 2000's

What Will We Talk About?

- Why is everyone still talking about NoSql?
- Why is everyone talking about Apache Cassandra?
- What makes Data Modeling for Apache Cassandra different?
- How to fix common mistakes

Who is Amanda?

WASHINGTON HUSKIES®

hp

Santa Clara University

Apache Trafodion

TERADATA ASTER

UDACITY

DATASTAX

Why NoSQL?

- Built to solve issues with Relational Databases
- All NoSQL Databases have in common
  - Made for Big Data
  - Horizontally scalable
  - High Throughput (low latency)
  - High Availability
- Issues of yesterday, we have in spades today!

Why Apache Cassandra?

- Many NoSQL Databases
- 10 years old -- it’s survived!
- Open Source with a strong community
- Built for the cloud
  - Cloud agnostic
- Masterless Architecture -- True High Availability

Why Apache Cassandra?

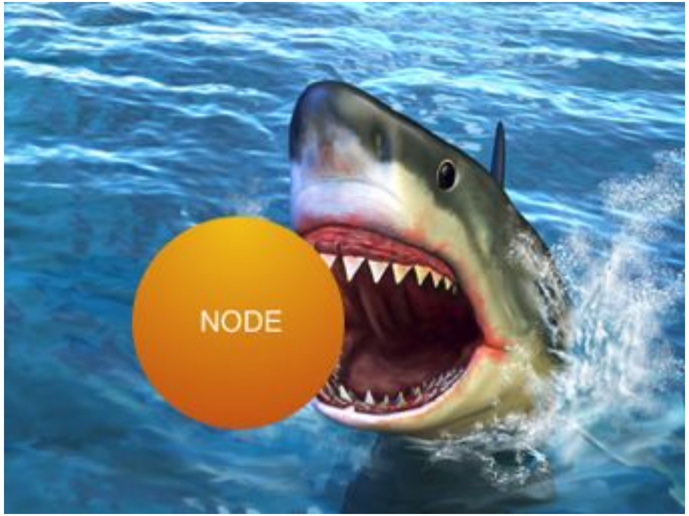
- Back-end of some of the most popular apps
  - Uber, Netflix, Hulu,Twitter, Apple
- Contributors to the code
  - DataStax, Apple, Twitter, Facebook

The Basics

- First developed by Facebook
- Top-level Apache Foundation project in 2010
- ***Distributed***, decentralized database
- Elastic scalability
  - Add/remove nodes with no downtime

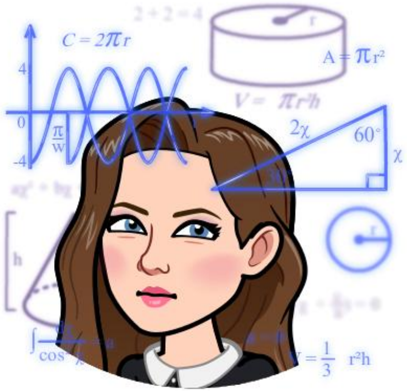
The Basics

- High performance
  - Very fast -- low latency
- High availability / fault tolerant
  - No single point of failure



Let’s Talk about the Big Topics

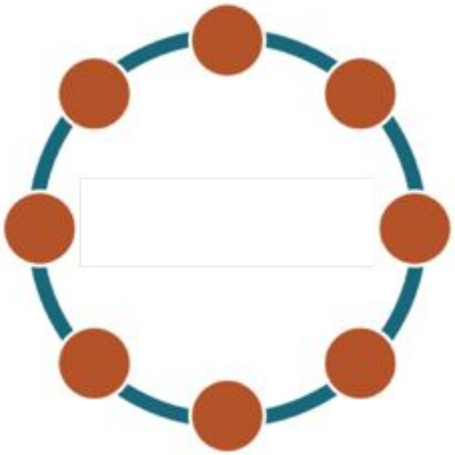
- Distributed Systems
- Replication
- Elastic Scalability
- High Availability
- Performance/Latency
  - Read Path
  - Write Path



★ Note: Don’t forget this is just a brief intro!

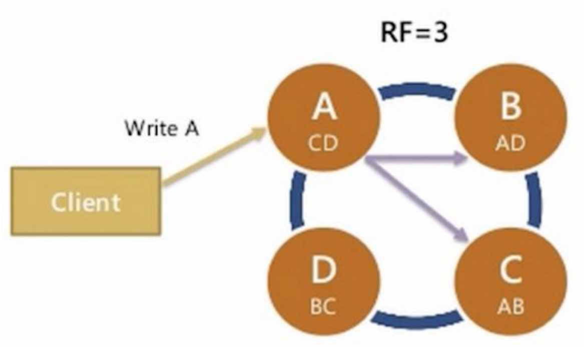
Distributed System

- Masterless Architecture
- All nodes have the same job
- All Clients can Connect to All Nodes
- All nodes ready for Read and Write
- Not all data on all nodes

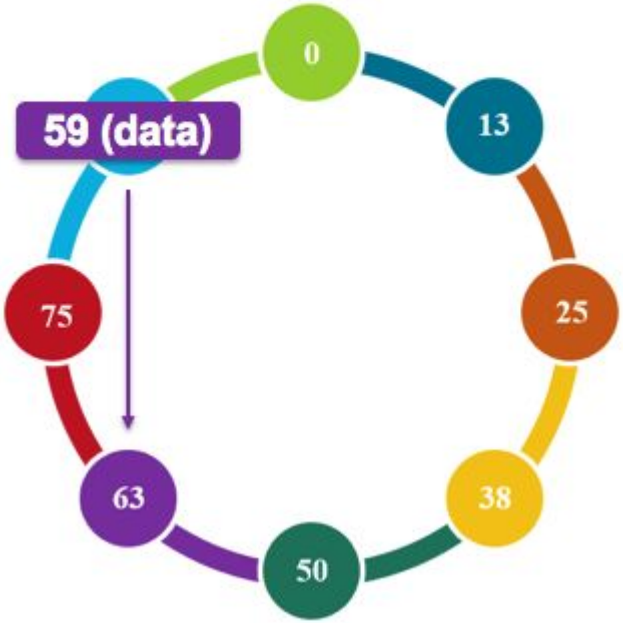


Replication

- Data must be copied to other nodes
  - High Availability
- Replication Factor (RF) is set by the user
  - 1 - # of Nodes
- Asynchronously replicated
  - Automatic
  - Peer-to-peer communication

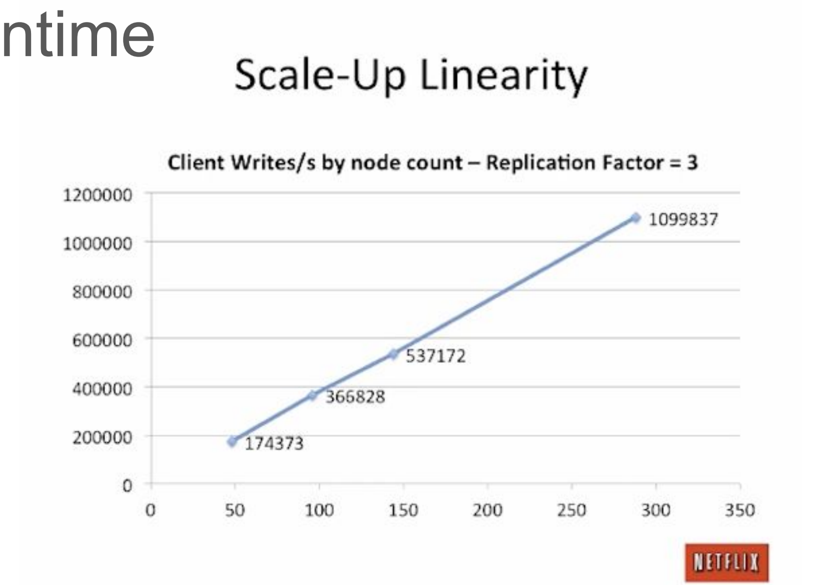


Token Range and The Ring



Elastic Scalability

- Nodes ++ = Performance ++
- Scale up or down with no downtime
  - Not even a restart!
- Reads and Writes both scale



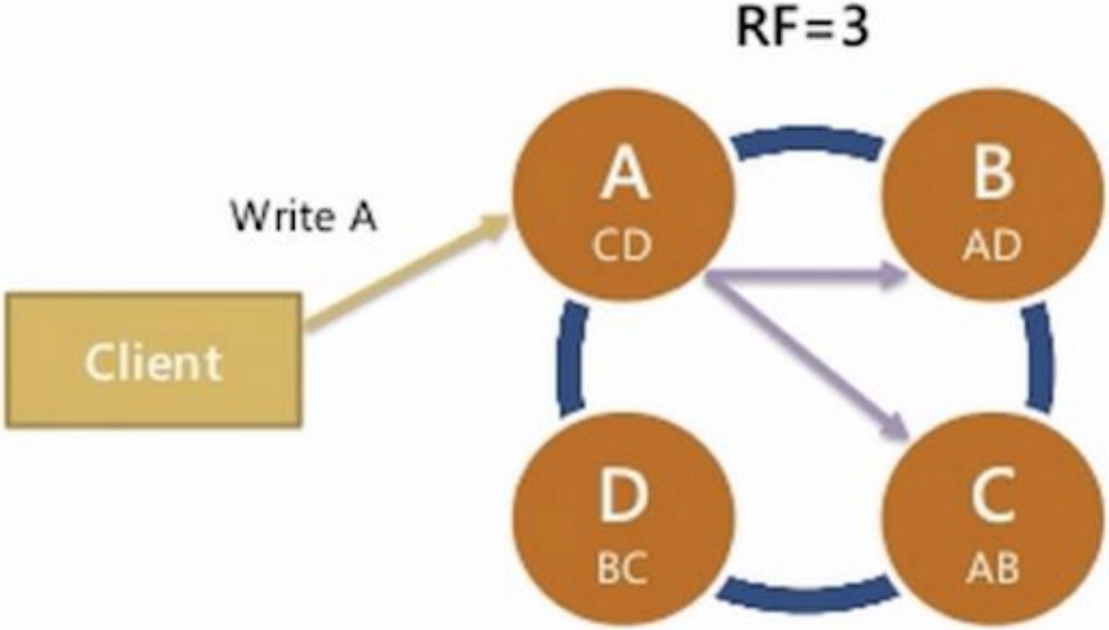
High Availability

- !Master node allows for high availability
  - No single point of failure
- Replication allows nodes to fail and data to still be available

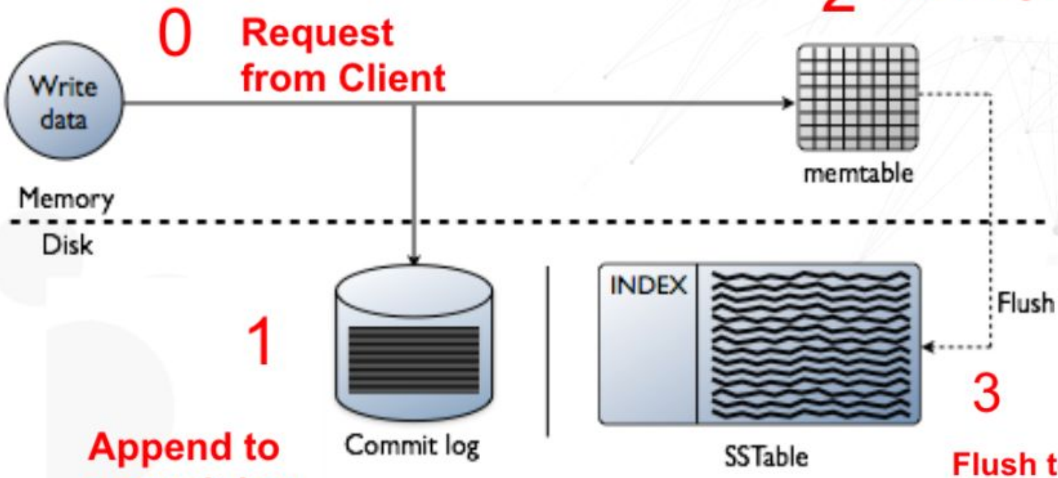
Latency

- How is low latency achieved?
- It’s all about the read and write path!
  - The write path is truly beautiful in its simplicity!

Write Path (Client to Cluster)



Write Path (Internal)



0 Request from Client

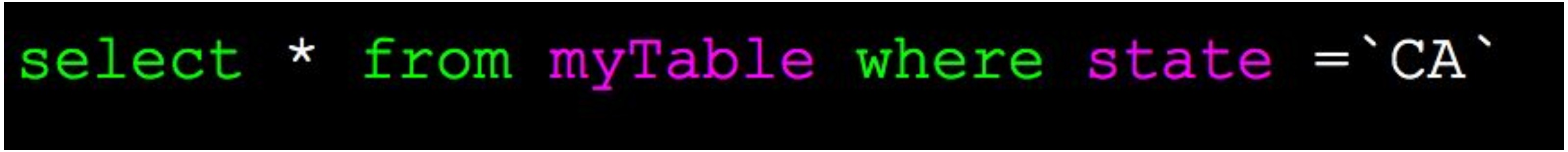
1 Append to commit log

2 Write to in memory table

3 Flush to persistent immutable table on disk

Read Path (Client to Cluster)

- Data modeling comes in to play here!
    - 1 “simple trick about Cassandra/Nosql”
- Partition data by nodes
- Will essentially query one node and return the data
    - Constant time READ access



```
select * from myTable where state =`CA`
```

Data Modeling is Different

RELATIONAL DATA MODEL != NoSQL DATA MODEL

CQL vs SQL

Data Modeling of the Past

- Normalization
  - Reduce Data Redundancy
  - Increase Data Integrity
  - 1970’s
  - 3rd Normal Form
  - Natural Process

JOINS will Save the Day --Maybe

- Combine Tables with JOINS
- Ad Hoc Queries are “okay”
- Expensive

Cartoon

iD	Name	City
0110	Duck Dodgers	Burbank
0100	Marvin	Mars

```
SELECT *
FROM cartoon
JOIN cartoon_job
ON
cartoon.id =
cartoon_job.id
```

Cartoon Job

iD	Occupation
0110	Astronaut
0100	Alien

What will Really Save the Day

- Demoralization
  - All about Performance (even in RDBMS)
  - Reducing the need for JOINS by combining/adding tables
  - Reducing time for reads
    - But will increase time for writes with data duplication

All About Cartoon

iD	Name	City	Job
0110	Duck Dodgers	Burbank	Astronaut
0100	Marvin	Mars	Alien

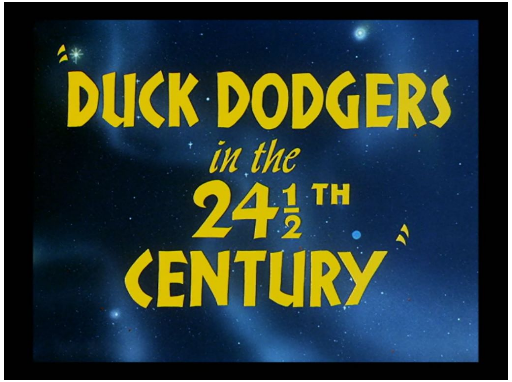
```
SELECT *
FROM
All_About_Cartoon;
```

Data Modeling in the 21st Century

- Start with your data
- Model the data according to normal forms
  - Reduce Data Redundancy
- Determine Queries based off of this data model

Data Modeling in the 24th 1/2 Century

- Flip it upside down!
- Start with queries/access patterns
- Create a denormalized data model
- Apply that model to your data



Denormalization** of tables in Apache Cassandra is **absolutely critical

Distributing the Data

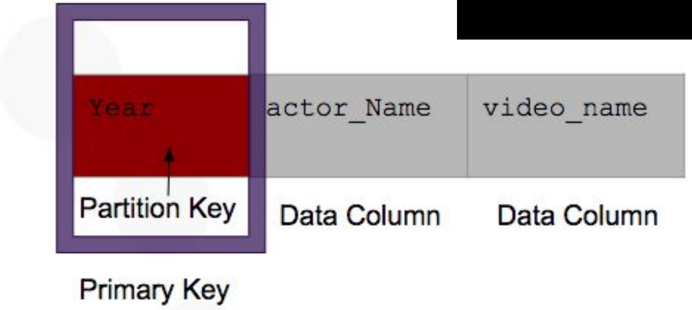
Primary Key

ALL about the PRIMARY KEY

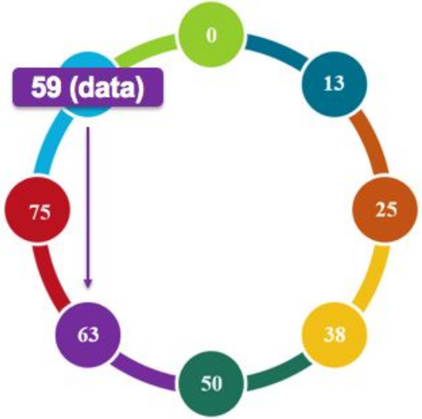
Primary Key

- Distributes the data
  - Partition Key
- Creates a Unique Key
- Made up of
  - Partition Key
  - 1 or more Clustering

```
CREATE TABLE
video_library
(year int,
actor_name text,
video_name text,
PRIMARY KEY (year))
```

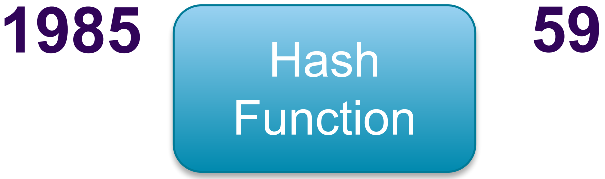


Partition Key



- Determine the distribution of data
- Partition key row value -- hashed
  - Stored on the node holds that range of values

```
INSERT INTO video_libary
(year, actor_name,
video_name) VALUES (1985,
'Michael J Fox', 'Back to
the Future')
```



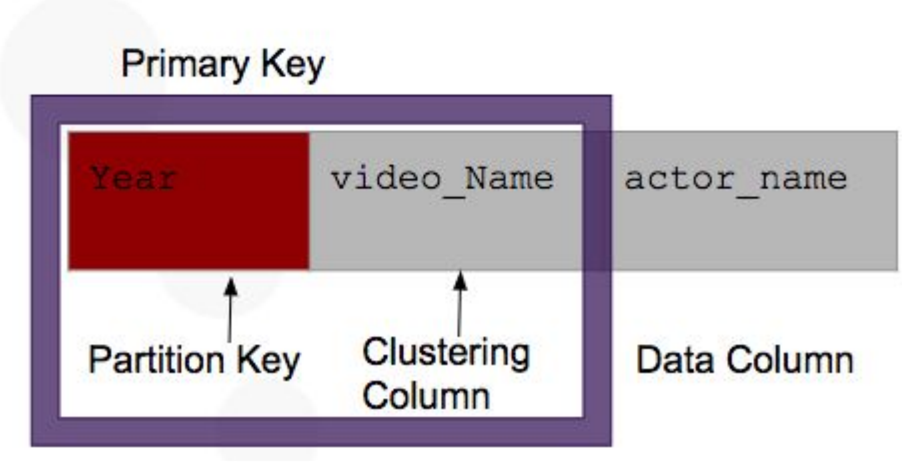
1985

Hash Function

59

Clustering Columns

- 0 or 1++ Clustering Columns
- Sort the data in sorted ascending order



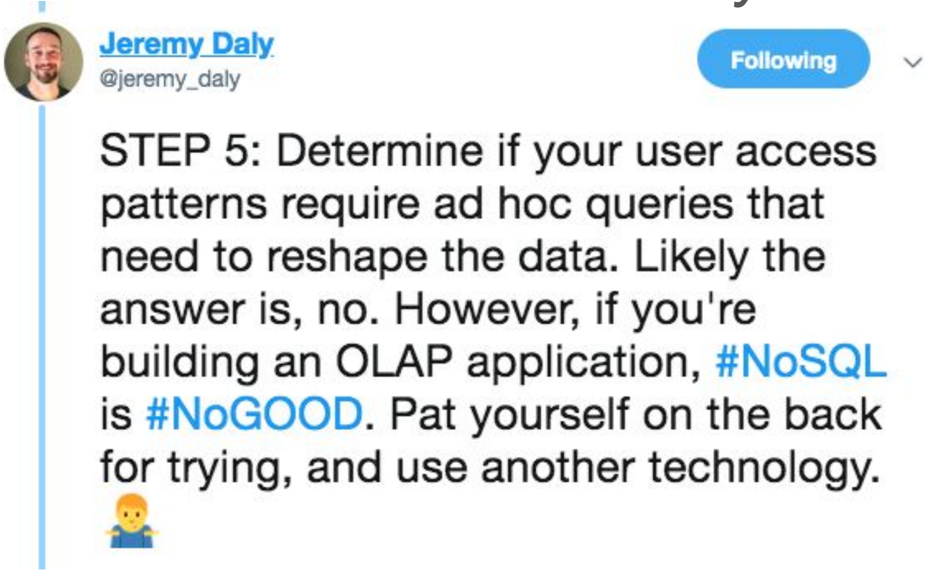
```
CREATE TABLE
video_library
(year int,
actor_name text,
video_name text,
PRIMARY KEY (year),
video_name)
```

Data Modeling Best Practices

- Do not move Relational Model **As-Is**
- 1 Table per 1 query
- Know Queries in Advance
  - Know your access patterns
- Think About your WHERE clause
- Denormalization is your Friend
  - You have money for more space, not for downtime

Data Modeling Best Practices

- *“How to Switch from RDBMS to DynamoDB in 20 steps”*



Jeremy Daly
@jeremy_daly

STEP 5: Determine if your user access patterns require ad hoc queries that need to reshape the data. Likely the answer is, no. However, if you're building an OLAP application, #NoSQL is #NoGOOD. Pat yourself on the back for trying, and use another technology. 🤷‍♂️

How to Fix Common Issues

- Data Distribution Issues
- Deleting Data Issues
- Data Integrity Issues

Issues with Primary Key

- Make sure Primary Key gives you unique rows
- Make sure Primary Key evenly distributes the data
  - Test and stress-test for Hot Partitions
- Make sure Partitions are not too large
  - Apache Cassandra 4.0 has a fix for this… but it’s not production ready
- Until then use Bucketing to break up Large Partitions

Tombstones

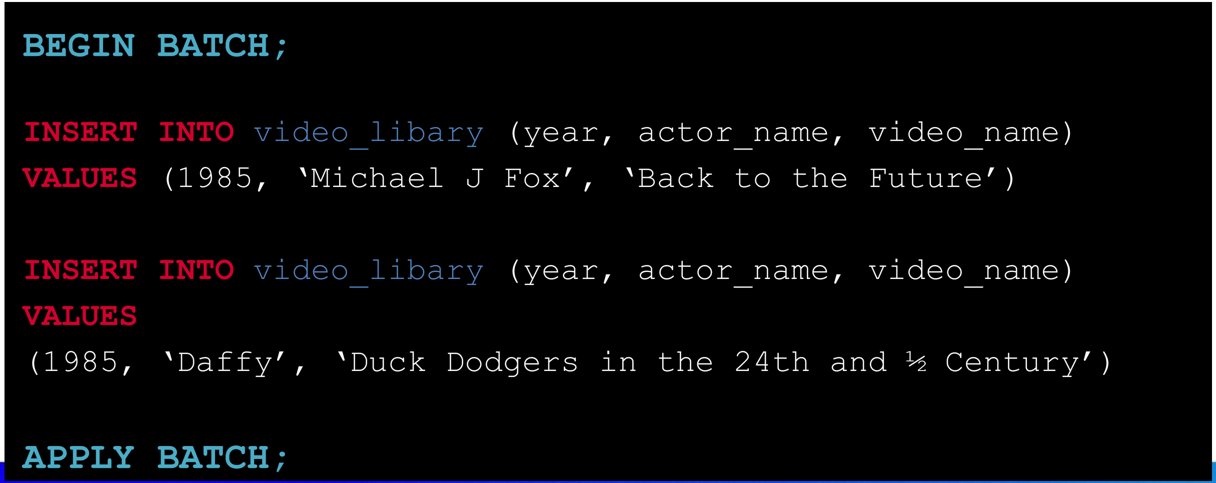
- Tombstone -- A deletion marker
  - Treats a delete as an additional insert or upsert
  - Will be removed with compaction
- When Do They Happen?
  - Deletes
  - Inserting `NULL` values
- Why so scary?
  - Impact performance -- slow reads
  - Disk space issues
  - This is why Apache Cassandra will crash!

Tombstones --Prevent

- Avoid Inserting NULL
- Inserting a new row if data column has no value leave it blank
- When editing a row (and using the drivers) can use `UNSET`
- Only update fields that you need to -- don’t add `NULL` … add nothing!

Batches

- Increase likelihood of having data integrity with batches
- Can put all inserts in one batch
- All will complete or an error will return after timeout



```
BEGIN BATCH;

INSERT INTO video_libary (year, actor_name, video_name)
VALUES (1985, ‘Michael J Fox’, ‘Back to the Future’)

INSERT INTO video_libary (year, actor_name, video_name)
VALUES
(1985, ‘Daffy’, ‘Duck Dodgers in the 24th and ½ Century’)

APPLY BATCH;
```

That was Awesome! Now what!

- Learn more about Cassandra: https://academy.datastax.com/
- Learn more about DataStax: https://www.datastax.com/
- Follow me on Twitter: @AmandaK_Data
- Slides will be on Github: https://github.com/amandamoran

DATASTAX ACADEMY

DataStax Cassandra D

Amanda

@AmandaK_Data

O'REILLY®
Velocity

THANK YOU

velocityconf.com/ca
#VelocityConf