

Week 2: Data Model

Creating an Efficient Data Model
for highly-loaded Applications



Cassandra Cloud-Native Workshop Series

Building Cloud-Native apps with Cassandra Expertise



~ 10,000 Registrations

<https://lime.link/blog/visualizing-crowd-sizes/>



The Crew



DataStax` Developer Advocacy Special Unit



Your hosts



Bettina Swynnerton

Community Engineer



Cédrick Lunven

Developer Advocate



David Jones-Gilardi

Developer Advocate



Aleksandr Volochnev

Developer Advocate



Jack Fryer

Community Manager





Your hosts



Aleksandr Volochnev 

Developer Advocate



Bettina Swynnerton 

Community Engineer



Jack Fryer 

Community Manager



Cédrick Lunven 

Developer Advocate



Erick Ramirez 

Lord of the Cassandra Rings



Cassandra Workshop SERIES

PART 1: How to build Applications with Cassandra

- Week 1 (7/01-7/08) : Getting Started with Cassandra ✓
- Week 2 (7/08-7/15) : Data Modelling for Apache Cassandra™ ○
- Week 3 (7/15-7/22) : Application Development, Backend Services and CRUD
- Week 4 (7/22-7/29) : Application Development, Microservices and REST API

PART 2: Test, Deploy and monitor your clusters and applications

- Week 5 (7/29-8/05): Operating your Cassandra clusters
- Week 6 (8/05-8/12): Running Cassandra performance tests
- Week 7 (8/12-8/19): Testing your deployments and troubleshooting
- Week 8 (8/19-8/26): Deploying Cassandra with Kubernetes



How does it work ?

1

**Attend one of the 2 LIVE STREAMED workshop
(Wednesday or Thursday) Choose the one what matches your
timezone.**

Courses: Datastax Developers @youtube

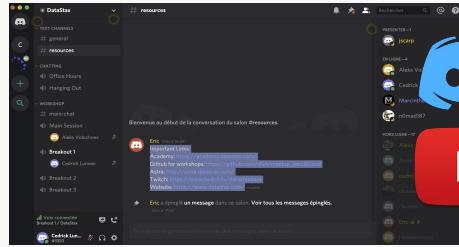


YouTube



Twitch

Questions: bit.ly/cassandra-workshop

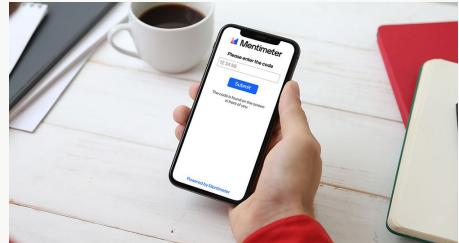


Discord



YouTube

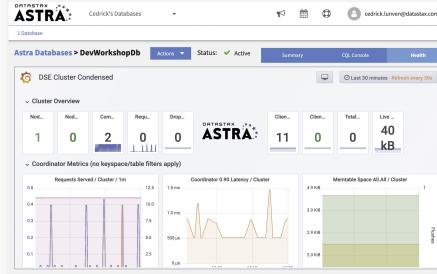
Quizz: menti.com



Available on iPhone
[App Store](#)

GET IT ON
[Google play](#)

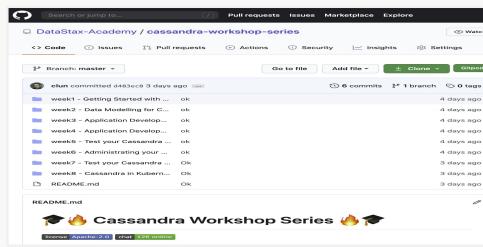
Runtime: astra.datastax.com



DATASTAX
ASTRA



Materials: github.com/DataStax-Academy



Coding (starting week #3)



How does it work ?

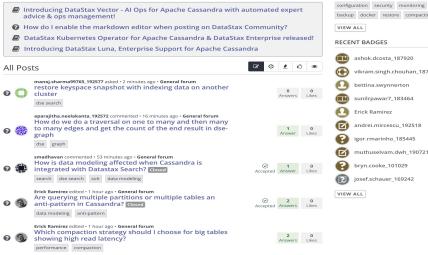


**Attend one of the 2 LIVE STREAMED workshop
(Wednesday or Thursday) -> They are recorded**

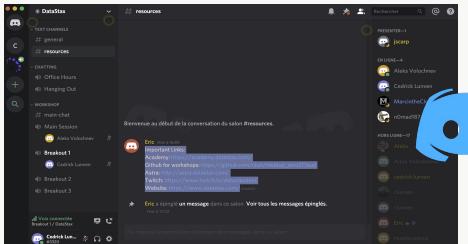


Do Homeworks

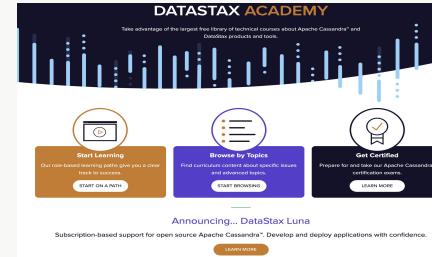
Forum: community.datastax.com



Chat: bit.ly/cassandra-workshop



Training: academy.datastax.com



Validate your week:

Questions Responses

Workshop Week 1 Validation Form

Form description

Fill the following form to validate your week. You can make it !

Cassandra Workshop Series
Your path to becoming a Cassandra expert!
From 1st July • 8 weeks • Live Teaching • Hands-on learning • Lots of fun interaction

1/3 - What is the email of Jeff at end "Working with CQL" notebook *



Google Forms

How does it work ?

- 
- 1
 - 2
 - 3

**Attend one of the 2 LIVE STREAMED workshop
(Wednesday or Thursday) -> They are recorded**

Homeworks

Relax.



Developer Workshop Series Week II

**What we will
cover:**

- Keyspaces, Tables, Partitions
- The Art of Data Modelling
- Data Types
- What's NEXT?

menti.com

89 49 47



Available on the iPhone
App Store

GET IT ON
Google play



Developer Workshop Series Week II

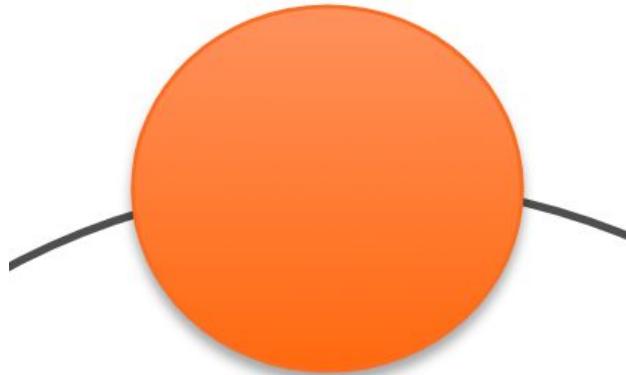
**What we will
cover:**

- Keyspaces, Tables, Partitions
- The Art of Data Modelling
- Data Types
- What's NEXT?

Infrastructure: a Node



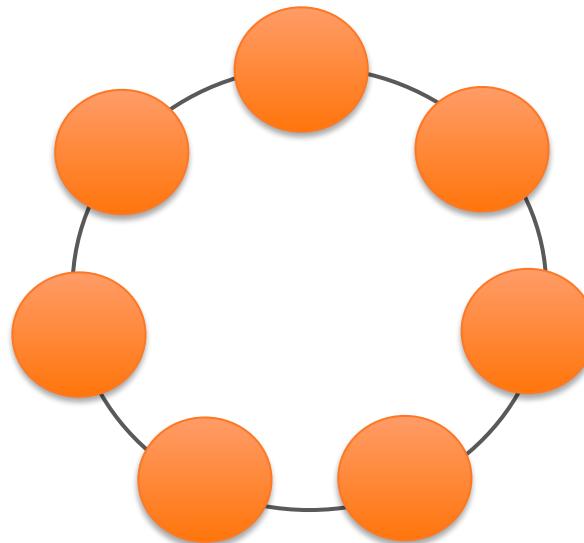
A single bare-metal server, a virtual instance or a docker container.



Infrastructure: a Datacenter (Ring)



A group of nodes located in the same physical location, a cloud datacenter or an availability zone.



Infrastructure: a Cluster



A group of
datacenters configured to
work together.

Data Structure: a Cell



An intersect of a row and
a column, stores data.



Data Structure: a Row



A single, structured data item in a table.

1	John	Doe	Wizardry
---	------	-----	----------

Data Structure: a Partition



A group of rows having the same partition token, a base unit of access in Cassandra.

IMPORTANT: stored together, all the rows are guaranteed to be neighbours.

ID	First Name	Last Name	Department
1	John	Doe	Wizardry
399	Marisha	Chapez	Wizardry
415	Maximus	Flavius	Wizardry

Data Structure: a Table



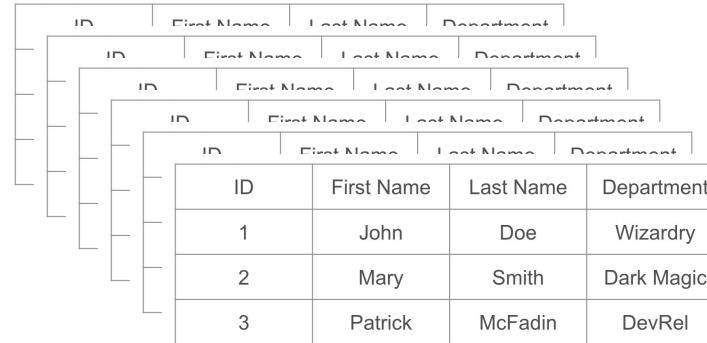
A group of columns and rows storing partitions.

ID	First Name	Last Name	Department
1	John	Doe	Wizardry
2	Mary	Smith	Dark Magic
3	Patrick	McFadin	DevRel

Data Structure: a Keyspace



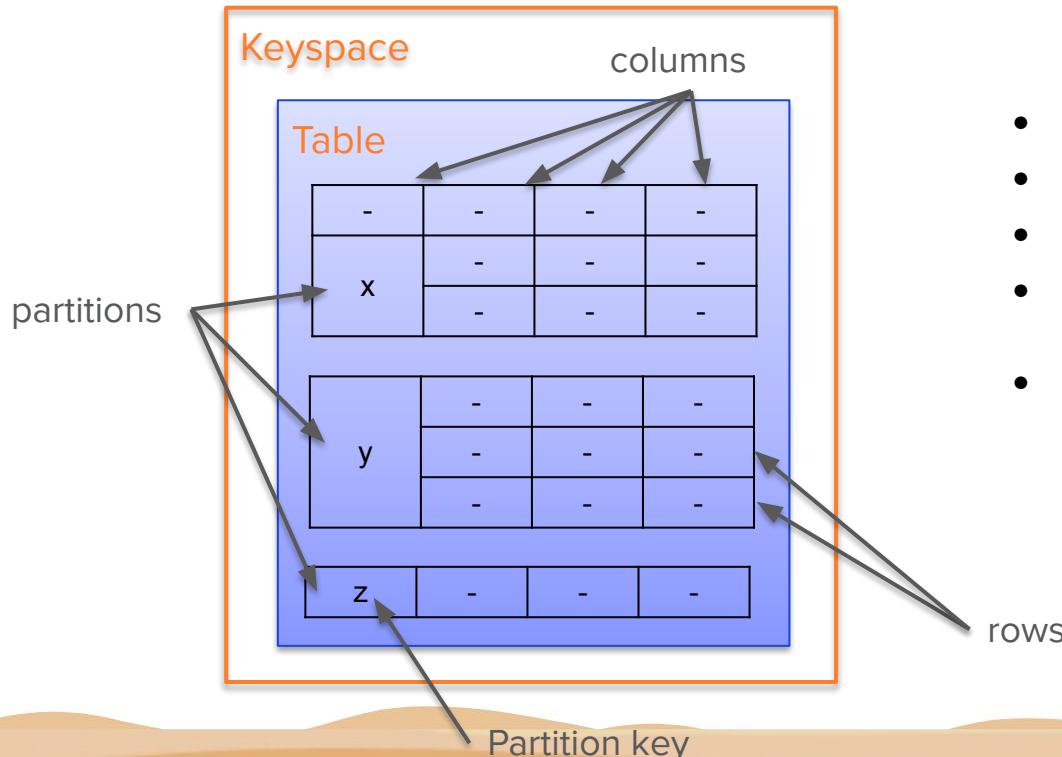
A group of tables sharing
replication strategy,
replication factor and other
properties



The diagram illustrates a Keyspace structure. It consists of four nested tables, each with columns: ID, First Name, Last Name, and Department. The innermost table contains three rows with data: ID 1 (John Doe, Wizardry), ID 2 (Mary Smith, Dark Magic), and ID 3 (Patrick McFadin, DevRel). Each row is connected by a vertical line to its corresponding row in the outer table, which in turn is connected to the outermost table's row.

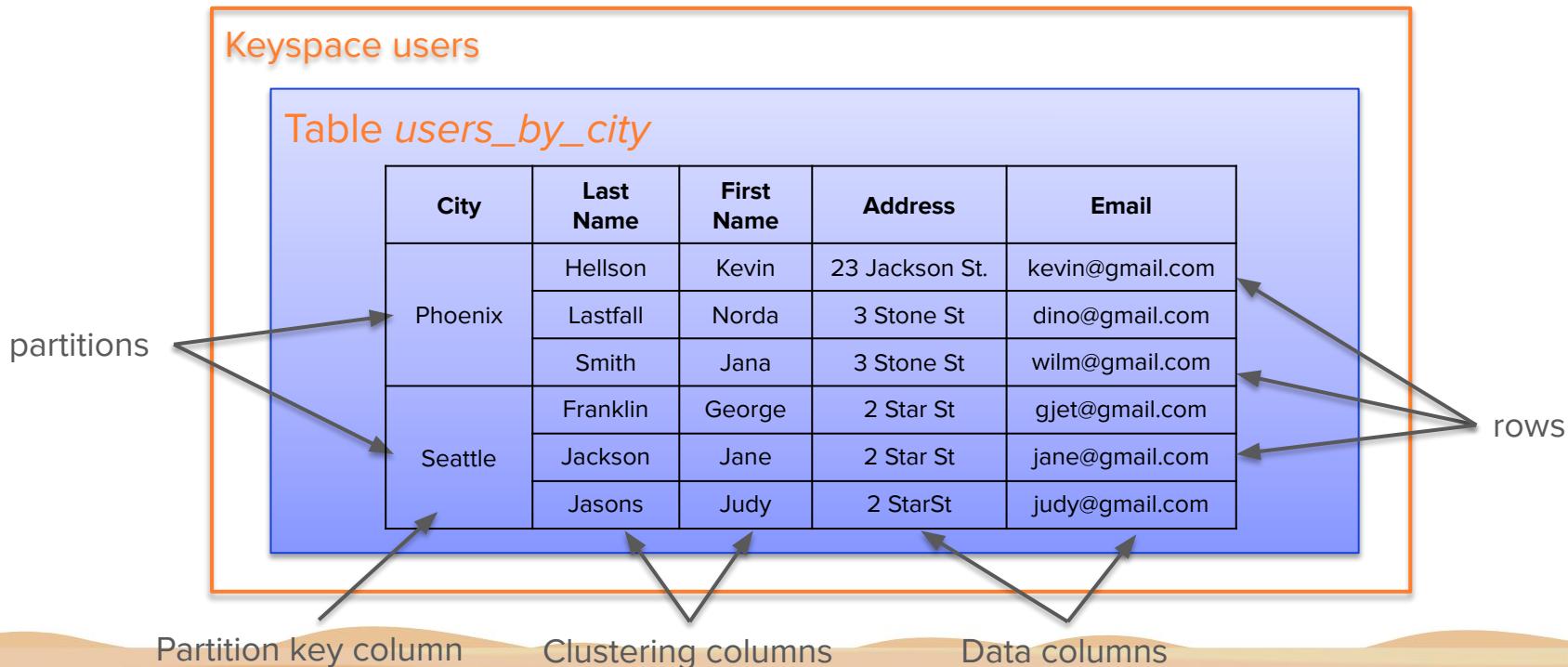
ID	First Name	Last Name	Department
1	John	Doe	Wizardry
2	Mary	Smith	Dark Magic
3	Patrick	McFadin	DevRel

Data Structure: Overall



- Tabular data model, with one twist
- *Keyspaces* contain *tables*
- *Tables* are organized in *rows* and *columns*
- Groups of related rows called *partitions* are stored together on the same node (or nodes)
- Each row contains a *partition key*
 - One or more columns that are hashed to determine which node(s) store that data

Example Data: Users organized by city



Creating a Keyspace in CQL

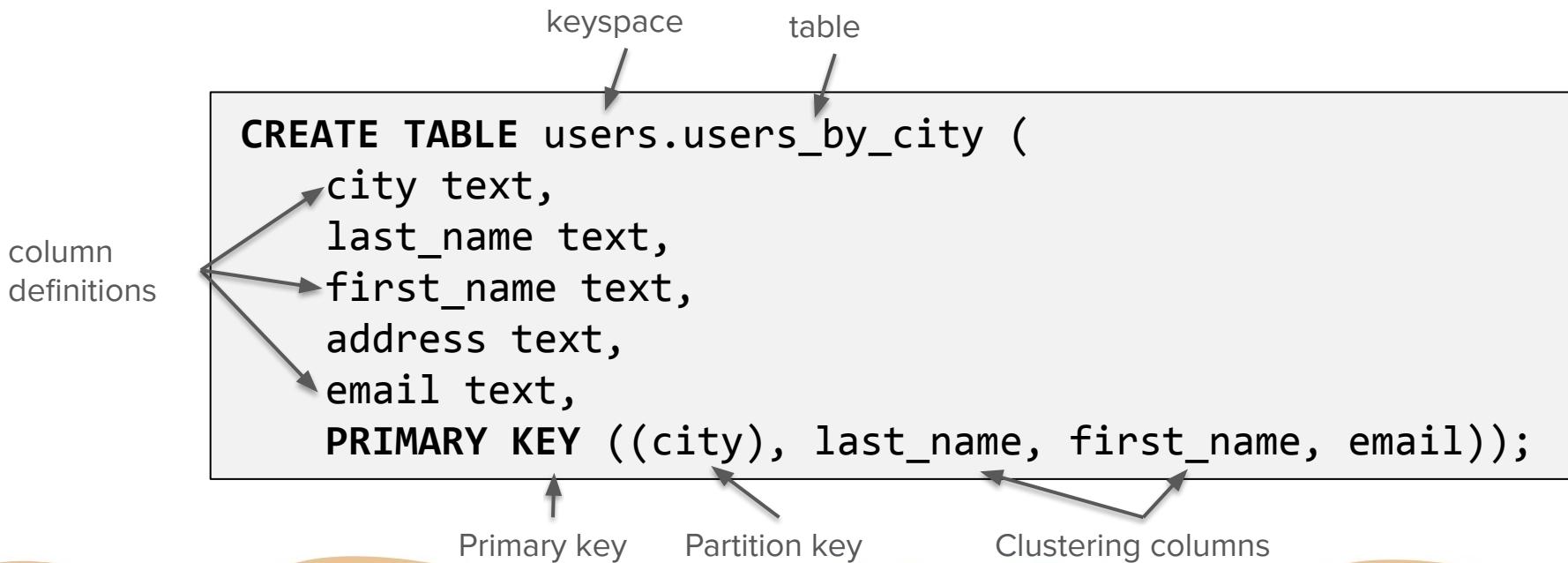
```
CREATE KEYSPACE users  
    WITH REPLICATION = {  
        'class' : 'NetworkTopologyStrategy',  
        'us-west-1' : 3,  
        'eu-central-1' : 3  
    };
```

keyspace

replication strategy

Replication factor by data center

Creating a Table in CQL



Primary Key

An identifier for a row. Consists of at least one Partition Key and zero or more Clustering Columns.

**MUST ENSURE UNIQUENESS.
MAY DEFINE SORTING.**

Good Examples:

```
PRIMARY KEY ((city), last_name, first_name, email);
```

```
PRIMARY KEY (user_id);
```

Bad Example:

```
PRIMARY KEY ((city), last_name, first_name);
```

```
CREATE TABLE users.users_by_city (
    city text,
    last_name text,
    first_name text,
    address text,
    email text,
    PRIMARY KEY ((city), last_name, first_name, email));
```



Partition Key

An identifier for a partition.

Consists of at least one column,
may have more if needed

PARTITIONS ROWS.

Good Examples:

```
PRIMARY KEY (user_id);
```

```
PRIMARY KEY ((video_id), comment_id);
```

Bad Example:

```
PRIMARY KEY ((sensor_id), logged_at);
```

```
CREATE TABLE users.users_by_city (
    city text,
    last_name text,
    first_name text,
    address text,
    email text,
    PRIMARY KEY ((city), last_name, first_name, email));
```

Partition key

Clustering columns

Clustering Column(s)

Used to ensure uniqueness and sorting order. Optional.

```
CREATE TABLE users.users_by_city (
    city text,
    last_name text,
    first_name text,
    address text,
    email text,
    PRIMARY KEY ((city), last_name, first_name, email));
```

Partition key

Clustering columns

PRIMARY KEY ((city), last_name, first_name);



Not Unique

PRIMARY KEY ((city), last_name, first_name, email);



PRIMARY KEY ((video_id), comment_id);



Not Sorted

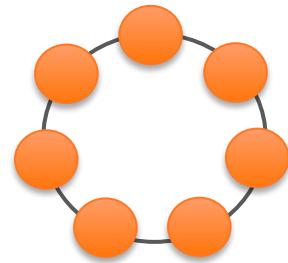
PRIMARY KEY ((video_id), created_at, comment_id);



How partition is happen to be?

```
CREATE TABLE users.users_by_city (
    city text,
    last_name text,
    first_name text,
    address text,
    email text,
    PRIMARY KEY ((city), last_name, first_name, email));
```

- Every node is responsible for a range of tokens (0-100500, 100501-201000...)
- INSERT a new row, we get the value of its Partition Key (can't be null!)
- We hash this value using MurMur3 hasher <http://murmurhash.shorelabs.com/>
“Seattle” becomes 2466717130 Partition Key = Seattle, Partition Token = 2466717130
- This partition belongs to the node[s] responsible for this token
- The INSERT query goes to the nodes storing this partition (Notice Replication Factor)



Rules of a Good Partition

The Slide of the Year Award!

- Store together what you retrieve together
- Avoid big partitions
- Avoid hot partitions

```
PRIMARY KEY (user_id);
```

```
PRIMARY KEY ((video_id), comment_id);
```

```
PRIMARY KEY ((country), user_id);
```



Rules of a Good Partition

The Slide of the Year Award!

- Store together what you retrieve together
- Avoid big partitions
- Avoid hot partitions

```
PRIMARY KEY ((video_id), created_at, comment_id);
```



```
PRIMARY KEY ((comment_id), created_at);
```



Rules of a Good Partition

The Slide of the Year Award!

- Store together what you retrieve together
- **Avoid big partitions**
- Avoid hot partitions

```
PRIMARY KEY ((video_id), created_at, comment_id);
```



```
PRIMARY KEY ((country), user_id);
```



Rules of a Good Partition

The Slide of the Year Award!

- Store together what you retrieve together
- Avoid big partitions
- **Avoid hot partitions**

```
PRIMARY KEY (user_id);
```



```
PRIMARY KEY ((video_id), created_at, comment_id);
```

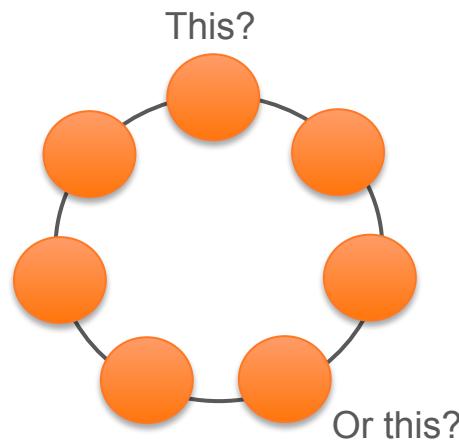


```
PRIMARY KEY ((country), user_id);
```



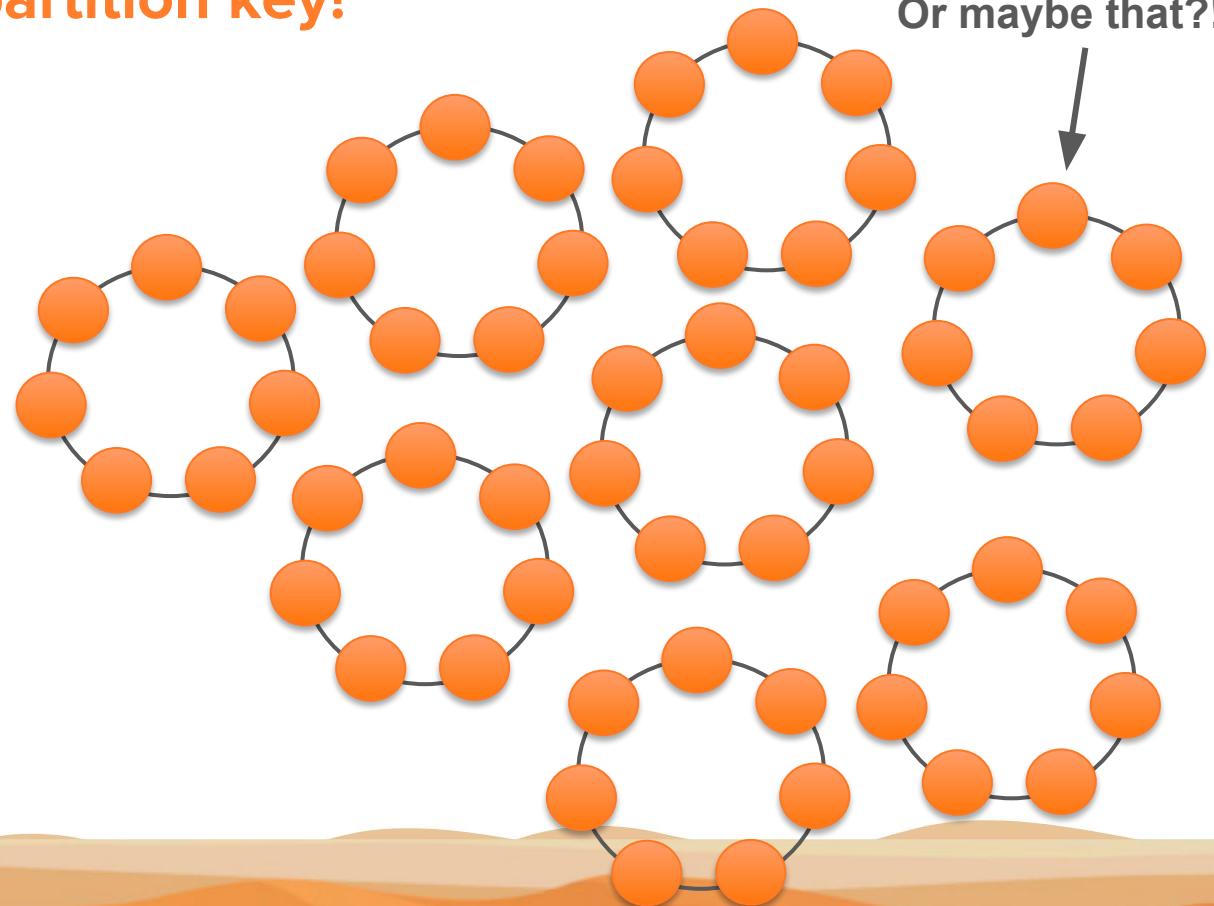
Always specify the partition key!

If there is no partition key in a query, which node you will ask?



Always specify the partition key!

If there is no partition key in a query, which node you will ask?



Always specify the partition key!

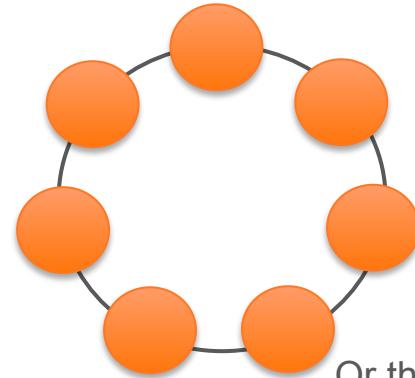
If there is no partition key in a query, which node you will ask?

```
CREATE TABLE users.users_by_city (
    city text,
    last_name text,
    first_name text,
    address text,
    email text,
    PRIMARY KEY ((city), last_name, first_name, email));
```

```
SELECT address FROM users_by_city WHERE first_name = "Anna";
```

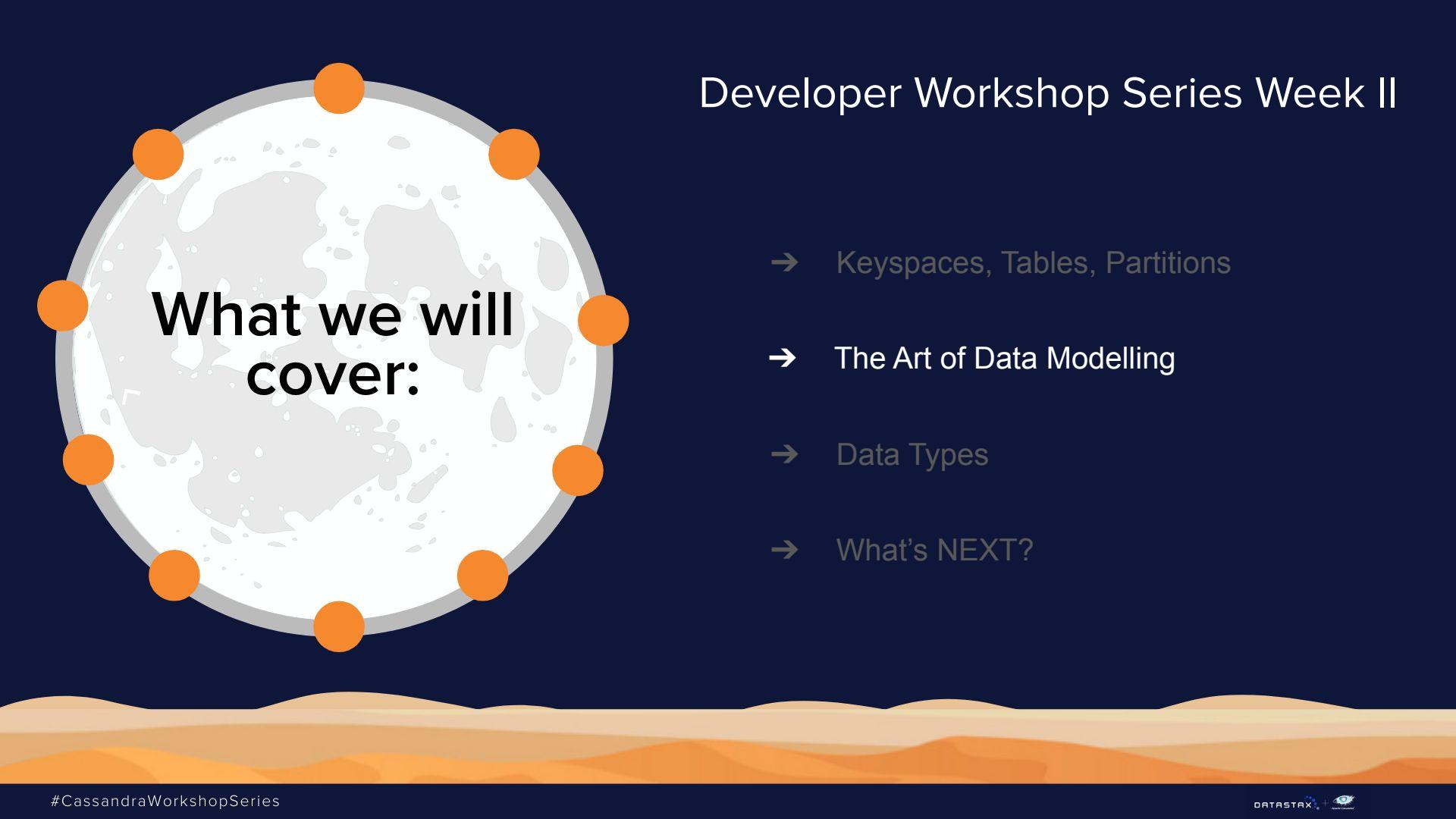
```
SELECT address FROM users_by_city WHERE city = "Otterberg" AND first_name = "Anna";
```

This?



Or this?





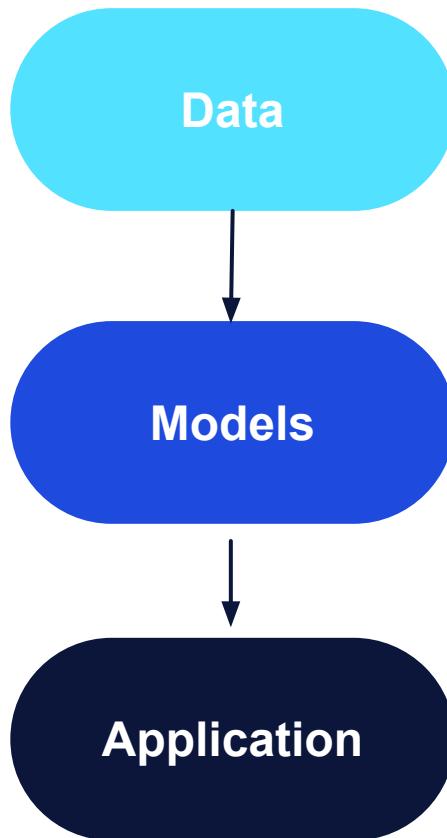
Developer Workshop Series Week II

**What we will
cover:**

- Keyspaces, Tables, Partitions
- The Art of Data Modelling
- Data Types
- What's NEXT?

Relational Data Modelling

1. Analyze raw data
2. Identify entities, their properties and relations
3. Design tables, using normalization and foreign keys.
4. Use JOIN when doing queries to join denormalized data from multiple tables



Employees		
userId	firstName	lastName
1	Edgar	Codd
2	Raymond	Boyce

Department	
departmentId	department
1	Engineering
2	Math

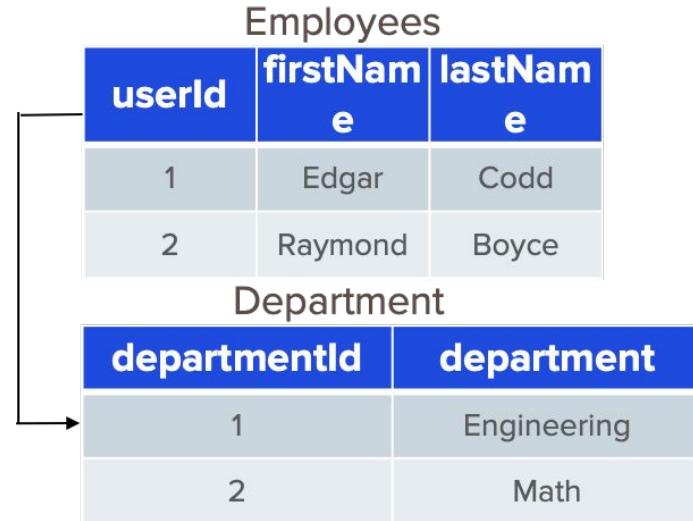


Normalization

“Database normalization is the process of structuring a relational database in accordance with a series of so-called normal forms in order to reduce data redundancy and improve data integrity. It was first proposed by Edgar F. Codd as part of his relational model.”

PROS: Simple write, Data Integrity

CONS: Slow read, Complex Queries



Employees		
userId	firstName	lastName
1	Edgar	Codd
2	Raymond	Boyce

Department	
departmentId	department
1	Engineering
2	Math

Denormalization

“Denormalization is a strategy used on a database to increase performance. In computing, denormalization is the process of trying to improve the read performance of a database, at the expense of losing some write performance, by adding redundant copies of data”

PROS: Quick Read, Simple Queries

CONS: Multiple Writes, Manual Integrity

Employees

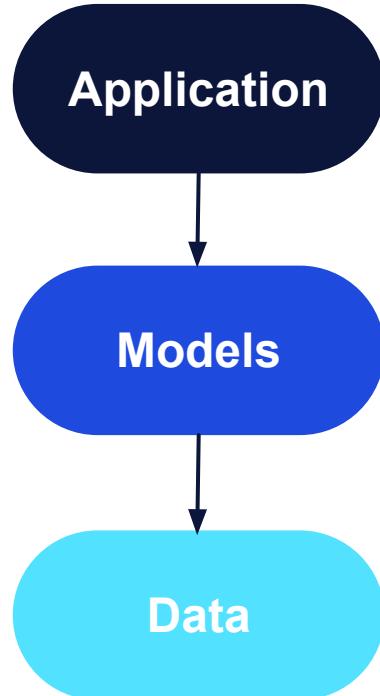
userId	firstName	lastName	department
1	Edgar	Codd	Engineering
2	Raymond	Boyce	Math

Department

departmentId	department
1	Engineering
2	Math

NoSQL Data Modelling

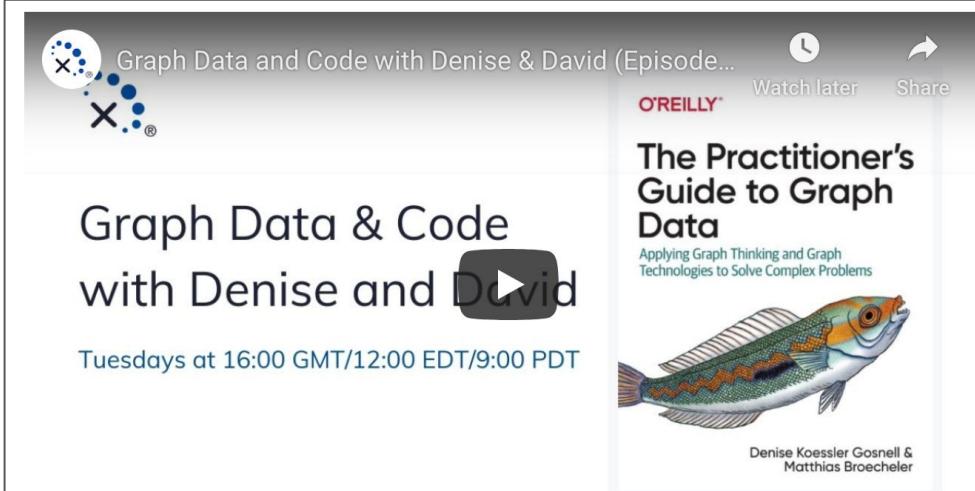
1. Analyze user behaviour (customer first!)
2. Identify workflows, their dependencies and needs
3. Define Queries to fulfill these workflows
4. Knowing the queries, design tables, using denormalization.
5. Use BATCH when inserting or updating denormalized data of multiple tables



id	firstName	lastName	department
1	Edgar	Codd	Engineering
2	Raymond	Boyce	Math



Let's go practical!



Graph Data and Code with Denise & David (Episode...)

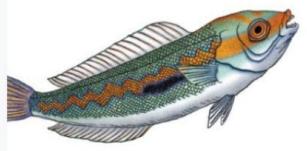
O'REILLY Watch later Share

Graph Data & Code with Denise and David

Tuesdays at 16:00 GMT/12:00 EDT/9:00 PDT

The Practitioner's Guide to Graph Data

Applying Graph Thinking and Graph Technologies to Solve Complex Problems



Denise Koessler Gosnell & Matthias Broeckeler

DATASTAX

0.0 overall rating

by: Adaline Walker

Share ▾

Added on 7/8/2020

Join Dr. Denise Gosnell as she continues to school David on their Graph journey from The Practitioner's Guide to Graph Data. This week we will:

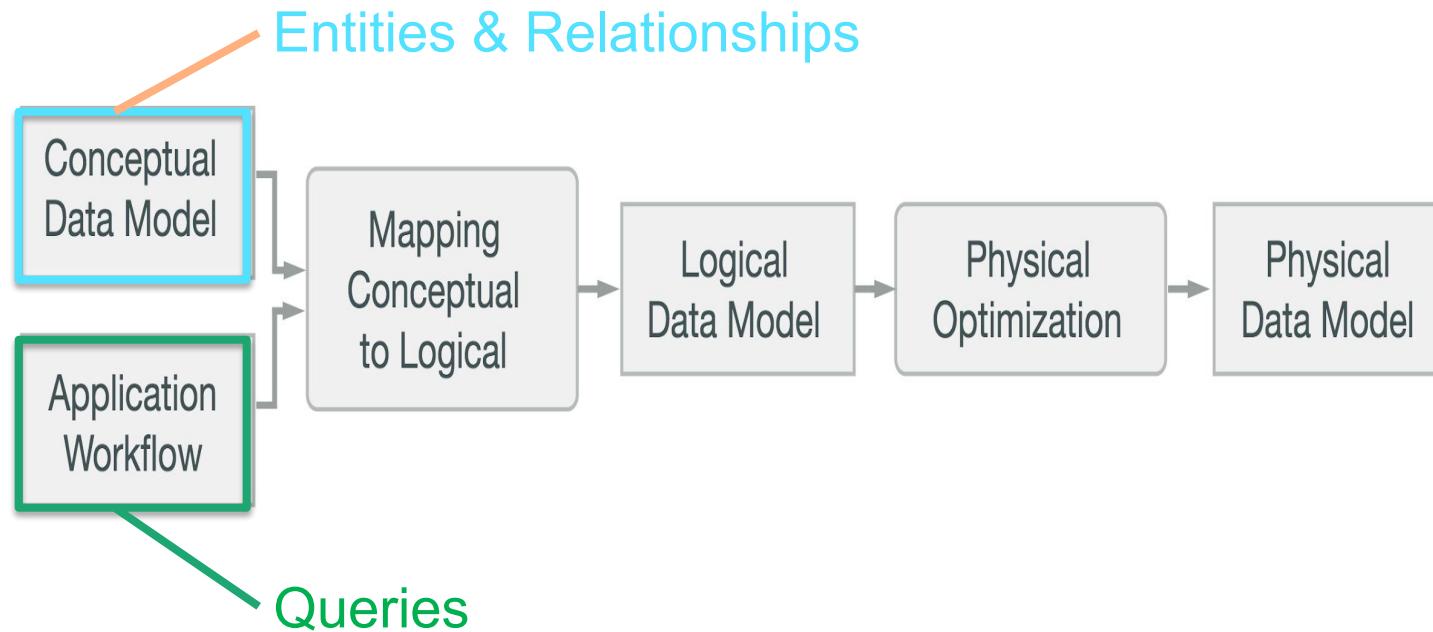
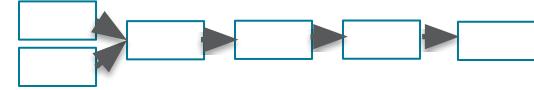
Show More

LATEST COMMENTS

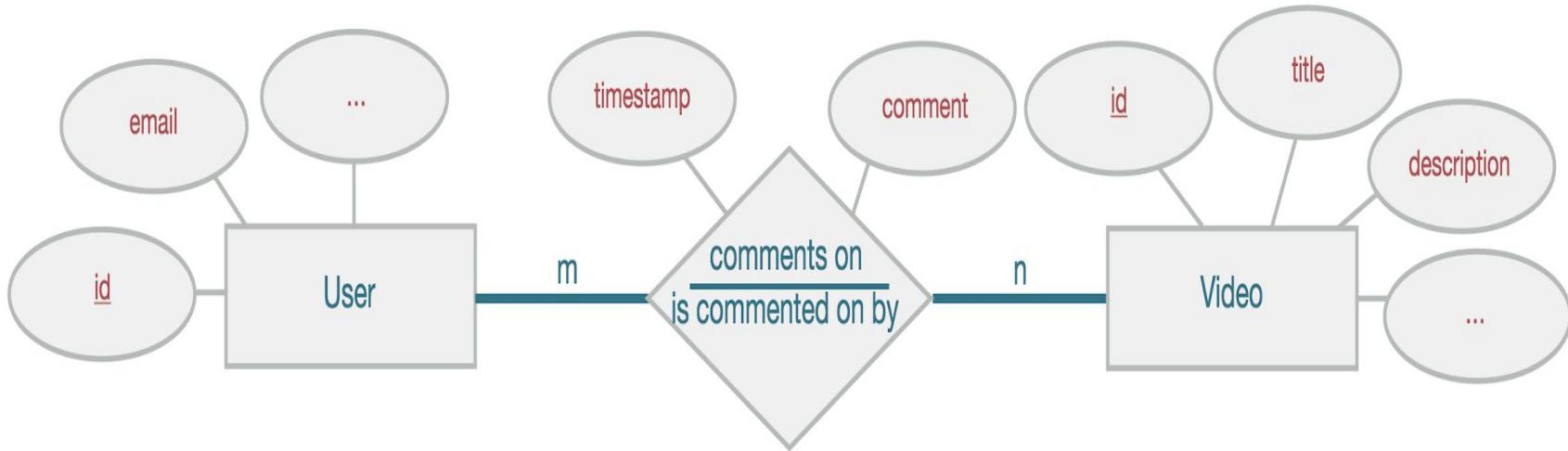
Aleksandr Volochnev a few seconds ago
Great video! Thank you!

Leave a comment

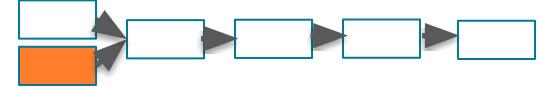
Designing Process: Step by Step



Designing Process: Conceptual Data Model



Designing Process: Application Workflow



Use-Case I:

- A User opens a Video Page

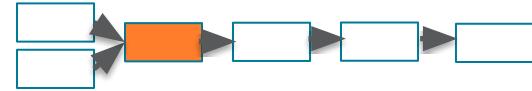
WF1: Find **comments** related to target **video** using its identifier, most recent first

Use-Case II + III:

- A User opens a Profile
- A Moderator verifies a User if spammer or not

WF2: Find **comments** related to target **user** using its identifier, get most recent first

Designing Process: Mapping



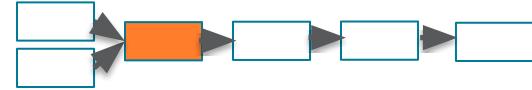
Query I: Find comments posted for a user with a known id (show most recent first)

→ `comments_by_user`

Query II: Find comments for a video with a known id (show most recent first)

→ `comments_by_video`

Designing Process: Mapping



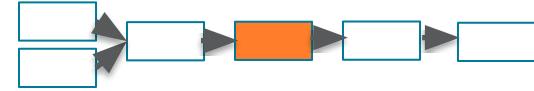
```
SELECT * FROM comments_by_user  
WHERE userid = <some UUID>
```

comments_by_user

```
SELECT * FROM comments_by_video  
WHERE videoid = <some UUID>
```

comments_by_video

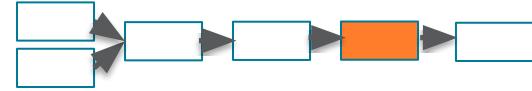
Designing Process: Logical Data Model



comments_by_user	
userid	K
creationdate	C ↓
commentid	C ↑
videoid	
comment	

comments_by_video	
videoid	K
creationdate	C ↓
commentid	C ↑
userid	
comment	

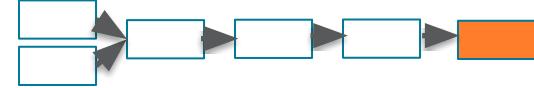
Designing Process: Physical Data Model



comments_by_user			
userid	UUID	K	
commentid	TIMEUUID	C	↓
videoid	UUID		
comment	TEXT		

comments_by_video			
videoid	UUID	K	
commentid	TIMEUUID	C	↓
userid	UUID		
comment	TEXT		

Designing Process: Schema DDL



```
CREATE TABLE IF NOT EXISTS comments_by_user (
    userid uuid,
    commentid timeuuid,
    videoid uuid,
    comment text,
    PRIMARY KEY ((userid), commentid)
) WITH CLUSTERING ORDER BY (commentid DESC);
```

```
CREATE TABLE IF NOT EXISTS comments_by_video (
    videoid uuid,
    commentid timeuuid,
    userid uuid,
    comment text,
    PRIMARY KEY ((videoid), commentid)
) WITH CLUSTERING ORDER BY (commentid DESC);
```

menti.com

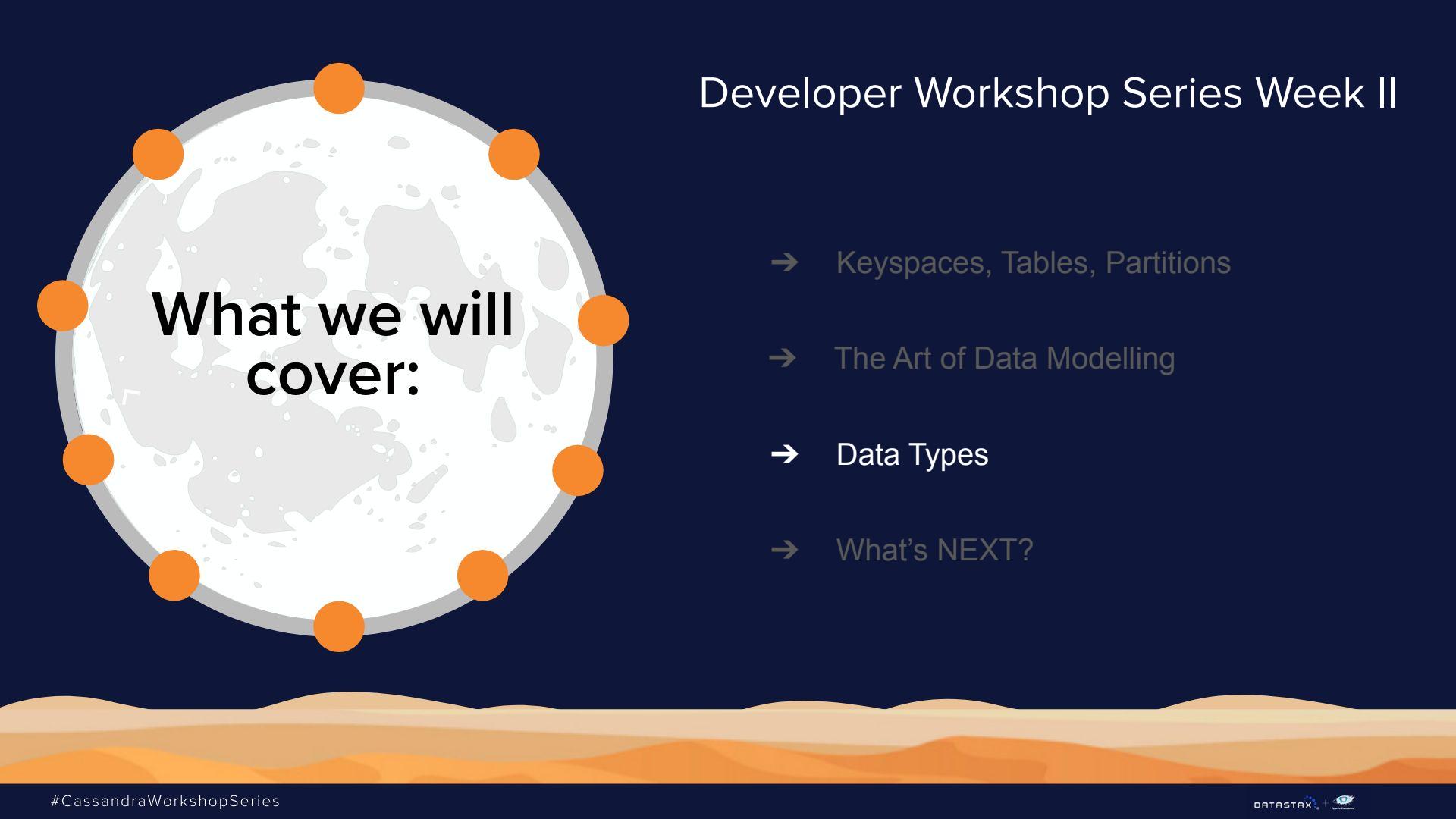
89 49 47



Available on the iPhone
App Store

GET IT ON
Google play

Developer Workshop Series Week II



What we will cover:

- Keyspaces, Tables, Partitions
- The Art of Data Modelling
- Data Types
- What's NEXT?

Basic Data Types

type	constants supported	description
ascii	string	ASCII character string
bigint	integer	64-bit signed long
blob	blob	Arbitrary bytes (no validation)
boolean	boolean	Either <code>true</code> or <code>false</code>
counter	integer	Counter column (64-bit signed value). See Counters for details
date	integer, string	A date (with no corresponding time value). See Working with dates below for details
decimal	integer, float	Variable-precision decimal
double	integer, float	64-bit IEEE-754 floating point
duration	duration, string	A duration with nanosecond precision. See Working with durations below for details
float	integer, float	32-bit IEEE-754 floating point
inet	string	An IP address, either IPv4 (4 bytes long) or IPv6 (16 bytes long). Note that there is no <code>inet</code> constant, IP address should be input as strings
int	integer	32-bit signed int
smallint	integer	16-bit signed int
text	string	UTF8 encoded string
time	integer, string	A time (with no corresponding date value) with nanosecond precision. See Working with times below for details
timestamp	integer, string	A timestamp (date and time) with millisecond precision. See Working with timestamps below for details
timeuuid	uuid	Version 1 UUID , generally used as a “conflict-free” timestamp. Also see Timeuuid functions
tinyint	integer	8-bit signed int
uuid	uuid	A UUID (of any version)
varchar	string	UTF8 encoded string
varint	integer	Arbitrary-precision integer

Collections



I'm an ordered LIST



I'm a MAP of key/value pairs

Key	Value
K1	V1
K2	V2
K3	V3
K4	V4
K5	V5

Collection: Set

```
CREATE killrvideo.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)
);
```

{‘Family’, ‘Disney’, ‘Princess’}

{‘Thriller’, ‘Short’}

{‘Tragicomedy’, ‘Western’}

Collection: Set

```
INSERT INTO killrvideo.videos (videoid, tags)  
VALUES(12345678-1234-1234-1234-123456789012,  
'Side-splitter', 'Short');
```

Insert

```
UPDATE killrvideo.videos  
SET tags = {'Dark', 'Sad'}  
WHERE videoid = 12345678-1234-1234-1234-123456789012;
```

Replace entire set

```
UPDATE killrvideo.videos  
SET tags = tags + {'Enthralling'}  
WHERE videoid = 12345678-1234-1234-1234-123456789012;
```

Add to set

Collection: List

```
CREATE killrvideo.actors_by_video (
    videoid uuid,
    actors list<text>, // alphabetical list of actors
    PRIMARY KEY (videoid)
);
```



Collection: List

```
INSERT INTO killrvideo.actors_by_video (videoid, actors)
VALUES(12345678-1234-1234-1234-123456789012,
      ['Adams', 'Baker', 'Cox']);
```

Insert

```
UPDATE killrvideo.actors_by_video
SET actors = ['Arthur', 'Beverly']
WHERE videoid = 12345678-1234-1234-1234-123456789012;
```

Replace entire list

```
UPDATE killrvideo.actors_by_video
SET actors = actors + ['Crawford']
WHERE videoid = 12345678-1234-1234-1234-123456789012;
```

Append

Collection: List

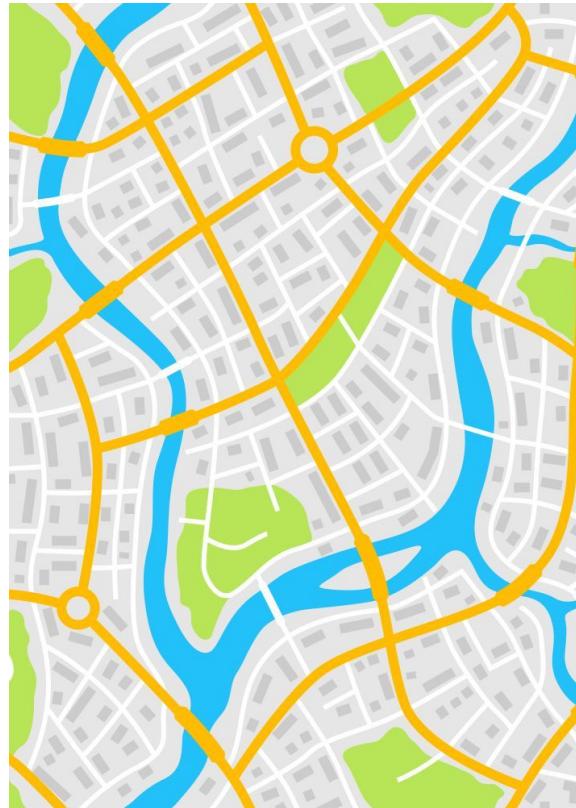
```
UPDATE killrvideo.actors_by_video  
    SET actors[1] = 'Brown'  
WHERE videoid = 12345678-1234-1234-1234-123456789012;
```

Replace an element

Note: replacing an element requires a read-before-write, which implies performance penalty.

Collection: Map

```
CREATE TABLE killrvideo.users(  
    userid        uuid,  
    phone_nos    map<text, text>,  
    PRIMARY KEY (userid)  
);
```



Collection: Map

```
INSERT INTO killrvideo.users (userid, phone_nos)
VALUES(12345678-1234-1234-1234-123456789012,
{'cell':'867-5309', 'home':'555-1212',
'busi':'800-555-1212'});
```

Insert

```
UPDATE killrvideo.users
SET phone_nos = {'cell':'867-5310', 'office':'555-1212'}
WHERE userid = 12345678-1234-1234-1234-123456789012;
```

Replace entire map

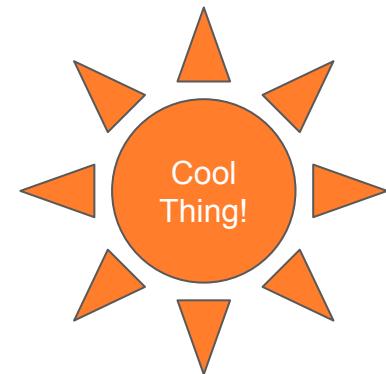
```
UPDATE killrvideo.users
SET phone_nos = phone_nos + {'desk': '270-555-1213'}
WHERE userid = 12345678-1234-1234-1234-123456789012;
```

Add to map

User Defined Types

```
CREATE TYPE killrvideo.address(
    street text,
    city   text,
    state  text,
);
```

```
CREATE TABLE killrvideo.users(
    userid      uuid,
    location    address,
    PRIMARY KEY (userid)
);
```



User Defined Types

```
INSERT INTO killrvideo.users (userid, location)  
VALUES(12345678-1234-1234-1234-123456789012,  
{street:'123 Main', city:'Metropolis', state:'CA'});
```

```
UPDATE killrvideo.users  
SET location = {street:'234 Elm', city:'NYC', state:'NY'}  
WHERE userid = 12345678-1234-1234-1234-123456789012;
```

```
UPDATE killrvideo.users  
SET location.city = 'Albany'  
WHERE userid = 12345678-1234-1234-1234-123456789012;
```

User Defined Types

```
SELECT location.city FROM killrvideo.users  
| WHERE userid = 12345678-1234-1234-1234-123456789012;
```

Select field

Counters

- 64-bit signed integer
- Use-case:
 - Imprecise values such as likes, views, etc.
- Two operations:
 - Increment
 - Decrement
 - First op assumes the value is zero

Counters

- Cannot be part of primary key
- Counters not mixed with other types in table
- Value cannot be set
- Rows with counters cannot be inserted
- Updates are not idempotent
 - Counters should *not* be used for precise values

Counters

```
CREATE TABLE killrvideo.video_playback_stats (
    videoid uuid,
    views counter,
    PRIMARY KEY (videoid)
);
```

Counters

Incrementing a counter:

```
UPDATE killrvideo.videos SET views = views + 1  
WHERE videoid = 12345678-1234-1234-1234-123456789012;
```

This format must be observed

This can be an integer value

Decrementing a counter:

```
UPDATE killrvideo.videos SET views = views - 1  
WHERE videoid = 12345678-1234-1234-1234-123456789012;
```

Just change the sign

menti.com

89 49 47



Available on the iPhone
App Store

GET IT ON
Google play

Developer Workshop Series Week II

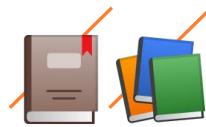


What we will cover:

- Keyspaces, Tables, Partitions
- The Art of Data Modelling
- Data Types
- What's NEXT?



Homework Week II



1. Do notebooks: **Data Modelling and Advanced Data Types**
<https://github.com/DataStax-Academy/cassandra-workshop-series/tree/master/week2-DataModel>
2. Watch the **DS220 Course** (videos only)
<https://academy.datastax.com/resources/ds220>
3. Validate your participation, fill **the form** (5 questions)
<https://forms.gle/YYZcZti4r3JgCLtV9>





Bonus Week II



- Single-Row Partitions
<https://katacoda.com/datastax/courses/cassandra-intro/tables-single-row-partitions>
- Multi-Row Partitions
<https://katacoda.com/datastax/courses/cassandra-intro/tables-multi-row-partitions>
- Advanced Data Types
<https://katacoda.com/datastax/courses/cassandra-intro/advanced-data-types>



Training Courses at DataStax Academy

- [DS201: Foundations of Apache Cassandra™](#)
- [DS220: Practical Application Data Modeling with Apache Cassandra™](#)



Developer Resources

LEARN

- Join academy.datastax.com
- Browse www.datastax.com/dev

ASK/SHARE

Join community.datastax.com

Ask/answer community user questions - share your expertise

CONNECT

Follow us

We are on Youtube - Twitter - Twitch!

MATERIALS

Slides and code for this course are available at

<https://github.com/DataStax-Academy/cassandra-workshop-series>

Part 1 - How to build Applications with Cassandra

Every Wednesday (NAM/LATAM) - 9am PDT / 12pm EDT / 5pm BST / 6pm CEST / 21:30pm IST

Every Thursday (APAC/EMEA) - 8am BST / 9am CEST / 12:30 pm IST / 15 pm SGT / 17pm ACT

July 1

Getting Started with Cassandra



July 8

Data Modelling for Apache Cassandra™



July 15

Application Development with Cassandra part 1

July 22

Application Development with Cassandra part 2

Part 2 - Test, Deploy and Monitor

Every Wednesday (NAM/LATAM) - 9am PDT / 12pm EDT / 5pm BST / 6pm CEST / 21:30pm IST

Every Thursday (APAC/EMEA) - 8am BST / 9am CEST / 12:30 pm IST / 15 pm SGT / 17pm ACT

July 29 Operating your Cassandra clusters

Aug 5 Running Cassandra performance tests

Aug 12 Testing your deployments and troubleshooting

Aug 19 Deploying Cassandra with Kubernetes

menti.com

89 49 47



Available on the iPhone
App Store

GET IT ON
Google play



Thank You

