

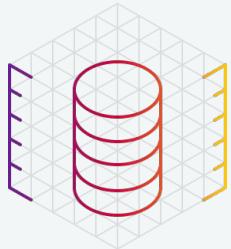
DATASTAX

DEVELOPERS

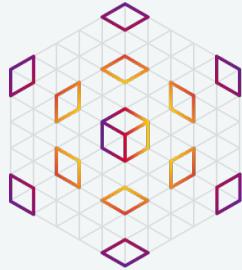
Application Development Bootcamp Week II: Designing Efficient Data Models

Building efficient applications with Apache Cassandra





Application Development Bootcamp



Learn about NoSQL
Databases

Jan 11, 2023

Scale your Apps
with efficient Data
Models

Jan 18, 2023

Build a Backend
with a NoSQL
Database

Jan 25, 2023



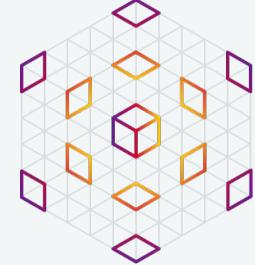
» Artem Chebotko

Developer Advocate



 @ArtemChebotko

 @ArtemChebotko



- ❖ Data professional, computer scientist
- ❖ Data modeling, data quality, data warehousing, data analytics
- ❖ Author of the Cassandra Data Modeling Methodology
- ❖ Google Cloud Certified Data Engineer

 Astra DB

 Astra STREAMING PULSAR



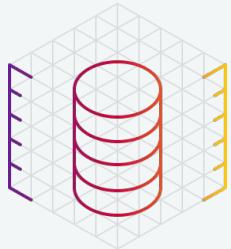
 APACHE Spark™

 cassandra

 Gremlin

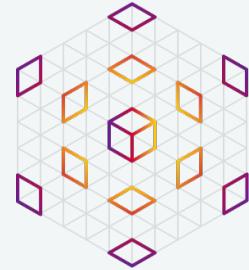
$G = (V, E)$



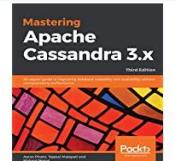
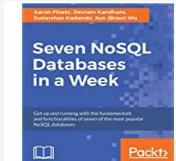


› Aaron Ploetz

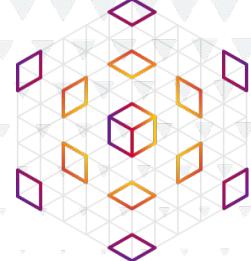
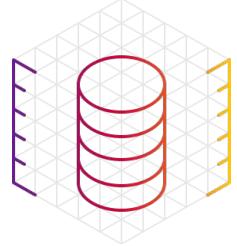
Developer Advocate



- ❖ DB Engineer, Author
- ❖ Former SWE/DevOps/DB Lead
@ W.W. Grainger & Target
- ❖ Worked as an author on:
 - Mastering Apache Cassandra 3.x
 - Seven NoSQL Databases in a Week



› Hands-on housekeeping



Source code + exercises + slides

IDE

```
StargateDemoApplication.java
1 package com.dataseven.demo.stargate;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class StargateDemoApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(StargateDemoApplication.class, args);
11     }
12 }
```



!github



!gitpod

Database + Api + Streaming



DATASTAX

ASTRA DB

!astradb

Nothing to install !



➤ Achievement unlocked!

[Issuers](#) / [DataStax Developers](#) / [Data Modelling Methodology Workshop](#) / Lai ██████████ @hotmail.com>



Data Modelling Methodology Workshop

Awarded to Lai ██████████ @hotmail.com)

Issued on **Jul 4, 2022** at 8:00 AM

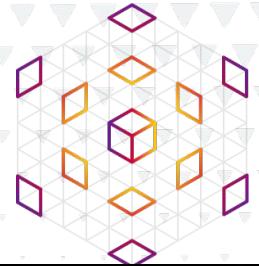
Upgrade Complete! This badge is to certify successful completion of the DataStax Cassandra Workshop: "Data Modelling Methodology for Apache Cassandra".

EARNING CRITERIA
Recipients must complete the earning criteria to earn this badge

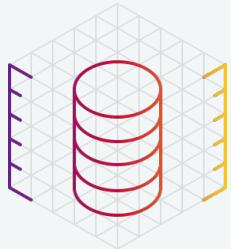
To earn this badge, individuals must complete the following steps during the [Data Modelling Methodology Workshop](#):

- Attend the lecture
- Complete the practical steps by doing all required exercises

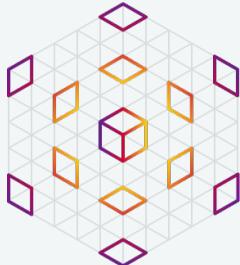
[View Badge Public Page](#) ↗



100+ and counting



» Agenda



01**Data Storage**

Data distribution & organization

02**Key Definition**

Primary, Partition, Clustering

03**CQL Data Types**

Brief review

04**Data Modeling**

Methodology, notation, tools

05**Optimization Techniques**

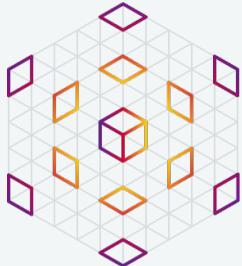
Five important optimizations

06**What's next?**

Homework, next sessions



» Agenda



01

Data Storage

Data distribution & organization

02

Key Definition

Primary, Partition, Clustering

03

CQL Data Types

Brief review

04

Data Modeling

Methodology, notation, tools

05

Optimization Techniques

Five important optimizations

06

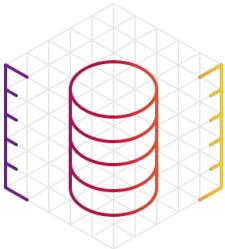
What's next?

Homework, next sessions

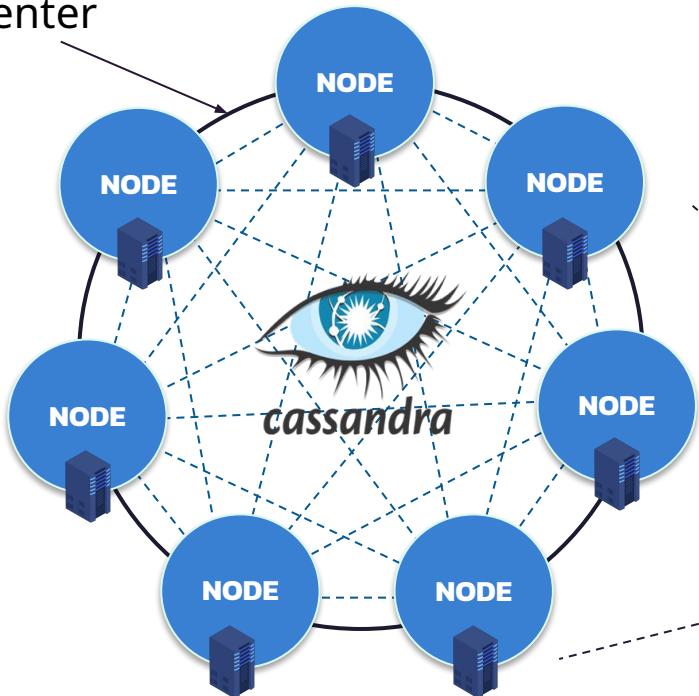


Data Distribution

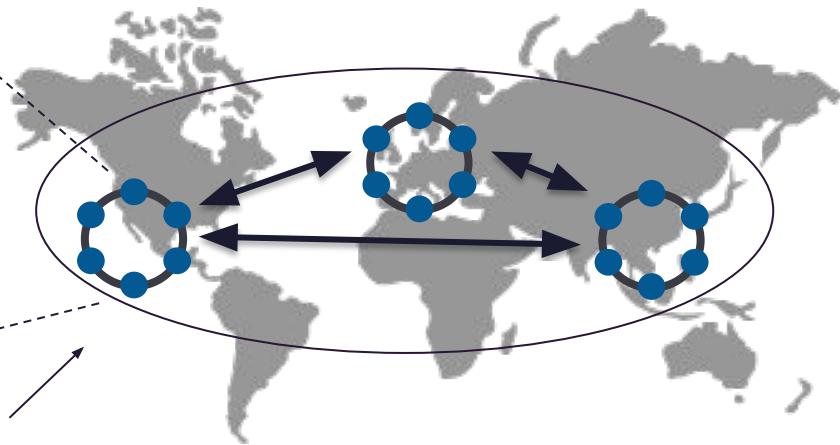
Architecture: cluster and datacenters

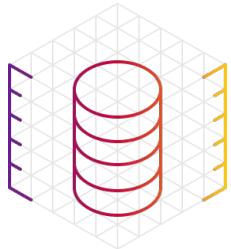


Datacenter

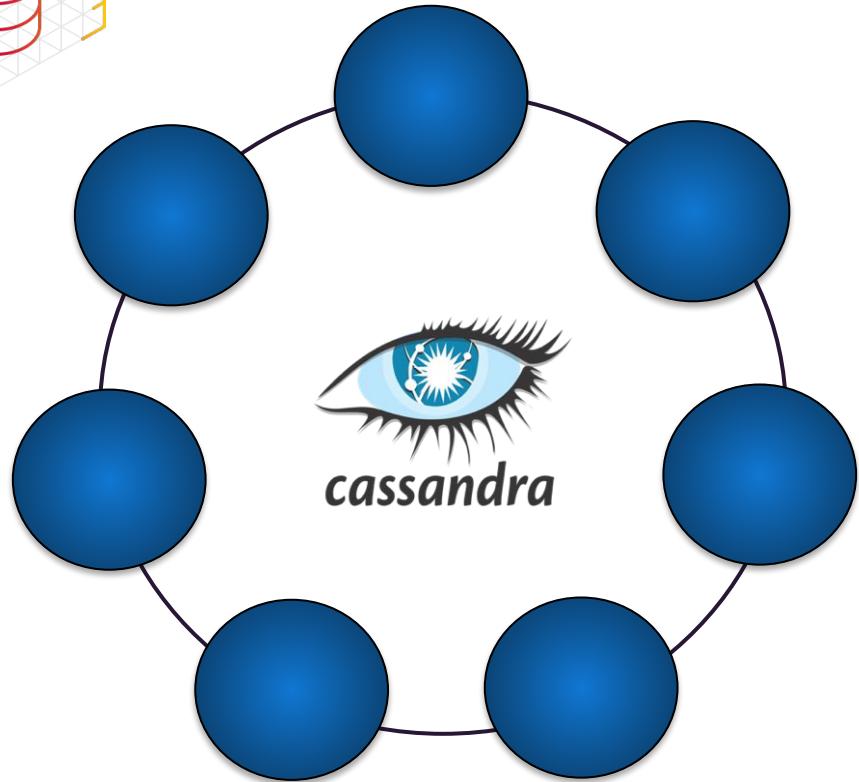
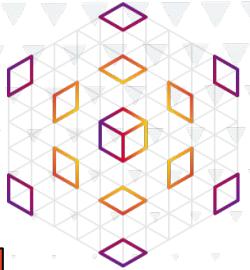


1. **NO Single Point of Failure**
2. Scales for writes and reads
3. Application can contact any node
(in case of failure - just contact next one)





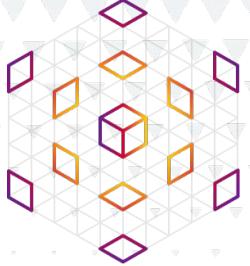
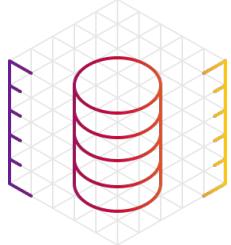
› Data is organized as tables



sensors_by_network		
network	sensor	temperature
forest	f001	92
	f002	88
volcano	v001	210
sea	s001	45
sea	s002	50
home	h001	72
car	c001	69
car	c002	70
dog	d001	40
road	r001	105
road	r002	110
ice	i001	35

Partition Key

Primary Key



› Data is organized as distributed tables

sensors_by_network		
network	sensor	temperature
forest	f001	92
forest	f002	88

forest	f001	92
forest	f002	88

sea	s001	45
sea	s002	50

car	c001	69
car	c002	70

volcano	v001	210
---------	------	-----



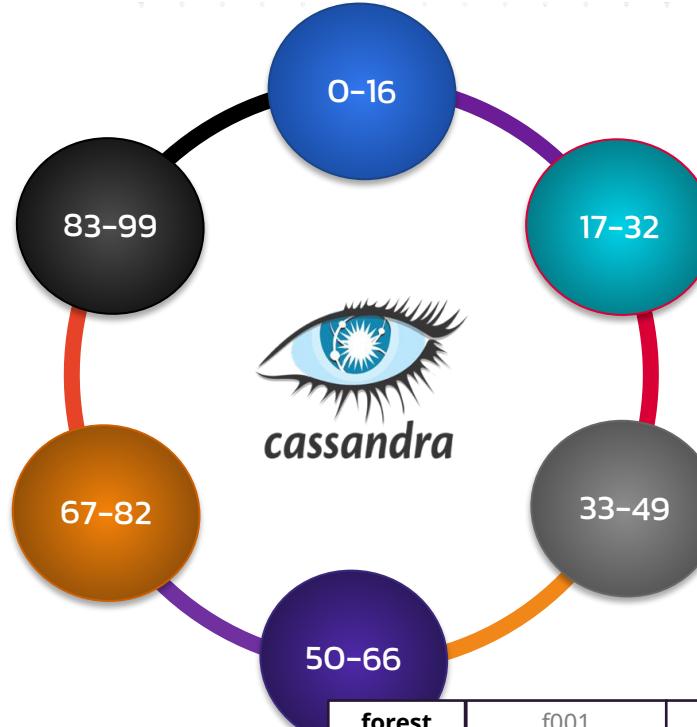
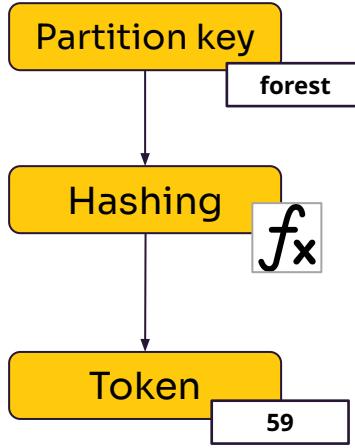
cassandra

home	h001	72
------	------	----

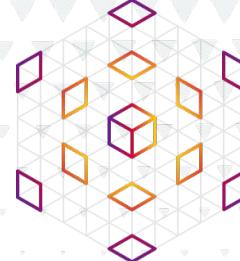
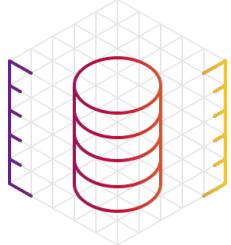
ice	i001	35
dog	d001	40

road	r001	105
road	r002	110

➤ Each node owns a range of tokens



forest	f001	92
forest	f002	88



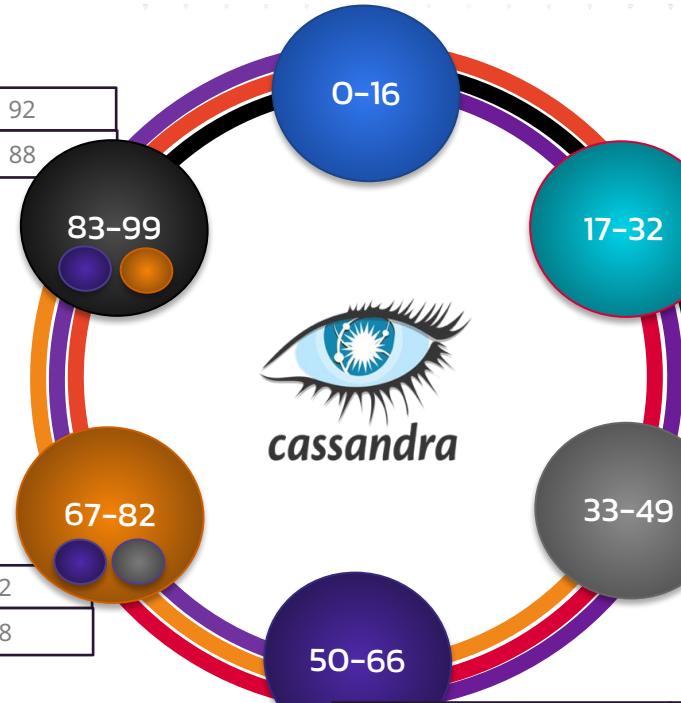
› Data is replicated according to a replication factor

RF = 3

forest	f001	92
forest	f002	88

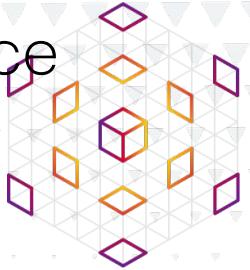
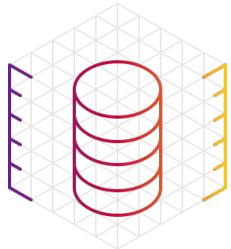
forest	f001	92
forest	f002	88

forest	f001	92
forest	f002	88

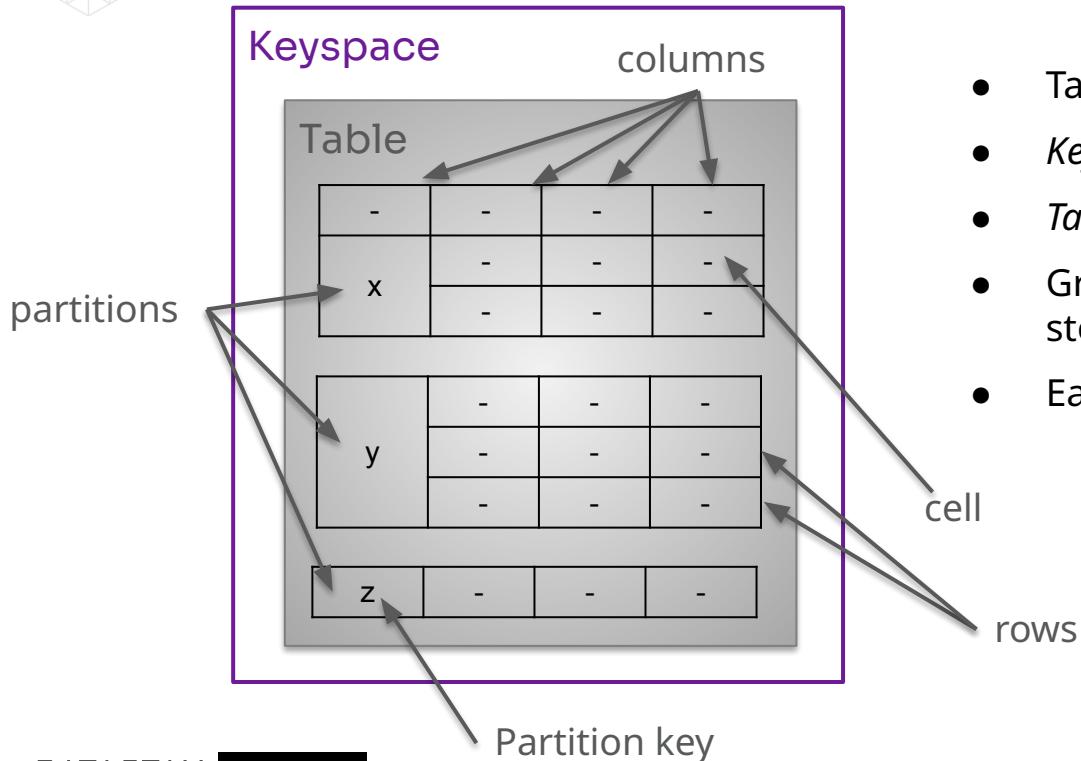




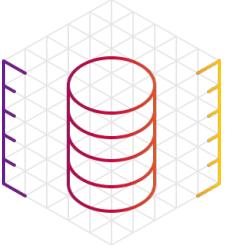
Data Organization



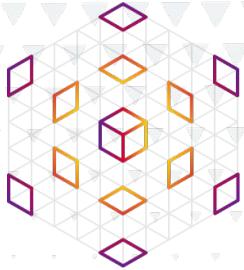
➤ Cell ∈ Row ∈ Partition ∈ Table ∈ Keyspace



- Tabular data model, with one twist
- *Keyspaces* contain *tables*
- *Tables* have *rows* and *columns*
- Groups of related rows called *partitions* are stored together on the same node (or nodes)
- Each row has a *partition key*



>Create a keyspace

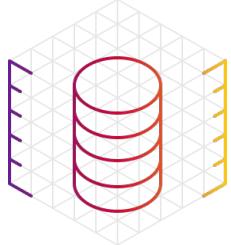


Keyspace

Replication strategy

```
CREATE KEYSPACE sensor_data  
WITH REPLICATION = {  
    'class' : 'NetworkTopologyStrategy',  
    'us-west-1' : 3,  
    'eu-central-1' : 5  
};
```

Replication factor by data center



>Create a table

```
CREATE TABLE sensor_data.sensors_by_network (
    network      text,
    sensor       text,
    temperature integer,
    PRIMARY KEY ((network), sensor)
);
```

Keyspace

Table

Primary key

Partition key

Clustering key



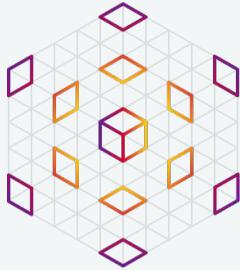
Lab 1

Create your Astra DB instance

github.com/datastaxdevs/workshop-cassandra-data-modeling



» Agenda



01

Data Storage

Data distribution & organization

02

Key Definition

Primary, Partition, Clustering

03

CQL Data Types

Brief review

04

Data Modeling

Methodology, notation, tools

05

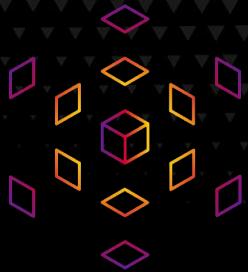
Optimization Techniques

Five important optimizations

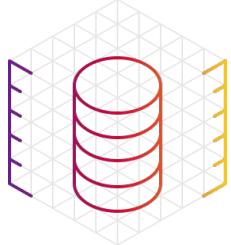
06

What's next?

Homework, next sessions



Key Definition



› Primary key defines how data is **stored**

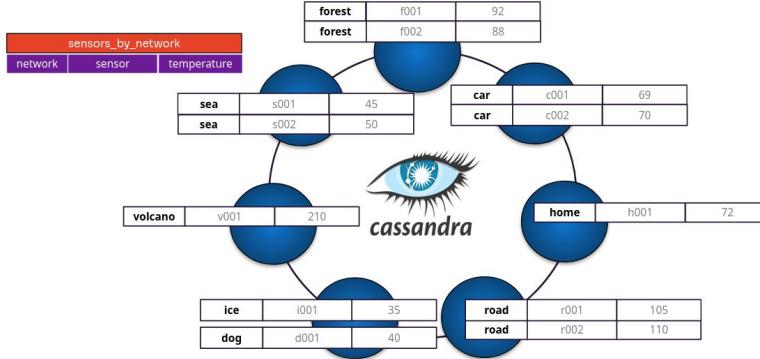
Primary Key = Partition Key + Clustering Key

Partition Key (mandatory)

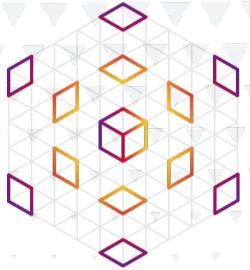
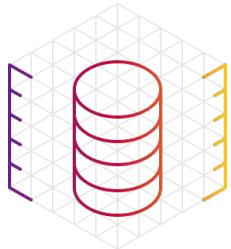
Defines data partitioning and distribution over the cluster

Clustering Key (optional)

Defines data uniqueness and ordering within a partition



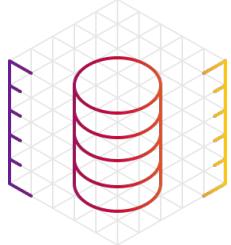
```
CREATE TABLE sensor_data.temperatures_by_sensor (
    sensor      TEXT,
    date        DATE,
    timestamp   TIMESTAMP,
    value       FLOAT,
    PRIMARY KEY ((sensor, date), timestamp)
) WITH CLUSTERING ORDER BY (timestamp DESC);
```



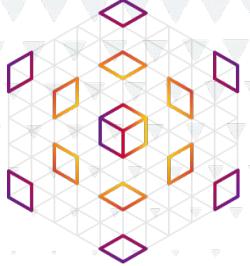
› Primary key defines how data is **retrieved**

```
PRIMARY KEY ((sensor, date), timestamp);
```

```
SELECT * FROM temperatures_by_sensor ...  
  
WHERE sensor = ?;  
  
WHERE sensor > ?;  
  
WHERE sensor = ? AND date > ?:  
  
WHERE sensor = ? AND date = ? AND timestamp > ?;
```



➤ Primary key definition cannot be **changed**



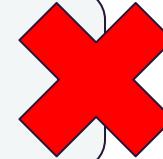
Important:

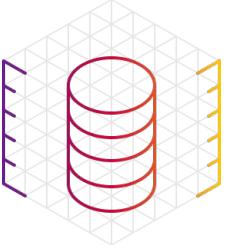
Once created, a primary key cannot be changed! You will need new tables and migration. Stay lazy, design it right in advance!

```
ALTER TABLE temperatures_by_sensor  
ADD season TEXT;
```



```
ALTER TABLE temperatures_by_sensor  
DROP PRIMARY KEY  
ADD PRIMARY KEY (sensor, timestamp)
```



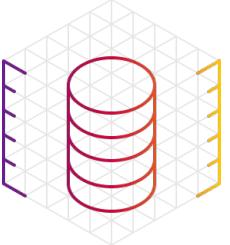


► Tables with single-row partitions



```
CREATE TABLE users (
    email      TEXT,
    name       TEXT,
    age        INT,
    date_joined DATE,
    PRIMARY KEY ((email))
);
```

```
CREATE TABLE movies (
    title      TEXT,
    year       INT,
    duration   INT,
    avg_rating FLOAT,
    PRIMARY KEY ((title, year))
);
```



› Tables with multi-row partitions



```
CREATE TABLE ratings_by_user (
    email  TEXT,
    year   INT,
    title  TEXT,
    rating INT,
    PRIMARY KEY ((email), year, title)
) WITH CLUSTERING ORDER BY
      (year DESC, title ASC);
```

```
CREATE TABLE ratings_by_movie (
    title  TEXT,
    year   INT,
    email  TEXT,
    rating INT,
    PRIMARY KEY ((title, year), email)
);
```



Lab 2

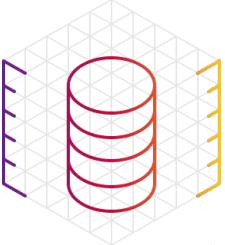
Tables with Single-Row and Multi-Row Partitions

github.com/datastaxdevs/workshop-cassandra-data-modeling

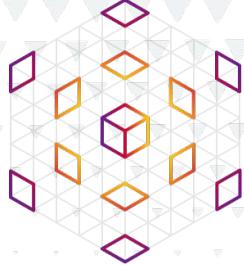


Rules for good partitioning





► Rules for good partitioning



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

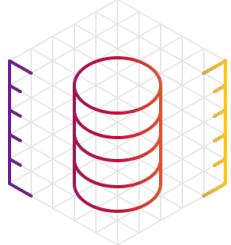
Q: Show temperature evolution over time for **sensor X** on **july 6th 2022**

```
PRIMARY KEY ((sensor, timestamp));
```

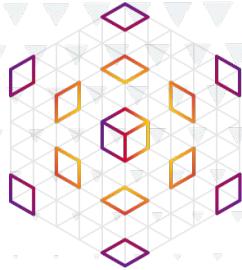


```
PRIMARY KEY (sensor, timestamp);
```





► Rules for good partitioning



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

```
PRIMARY KEY ((sensor), timestamp);
```

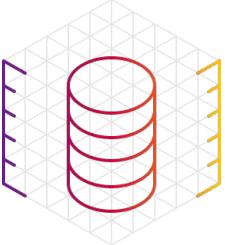


```
PRIMARY KEY ((sensor, date), timestamp);
```

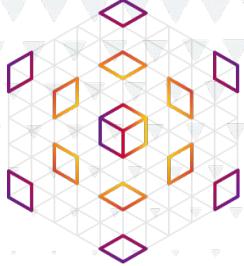


BUCKETING

- Up to 2 billion cells per partition
- Up to ~100k values in a partition
- Up to ~100MB in a Partition



► Rules for good partitioning



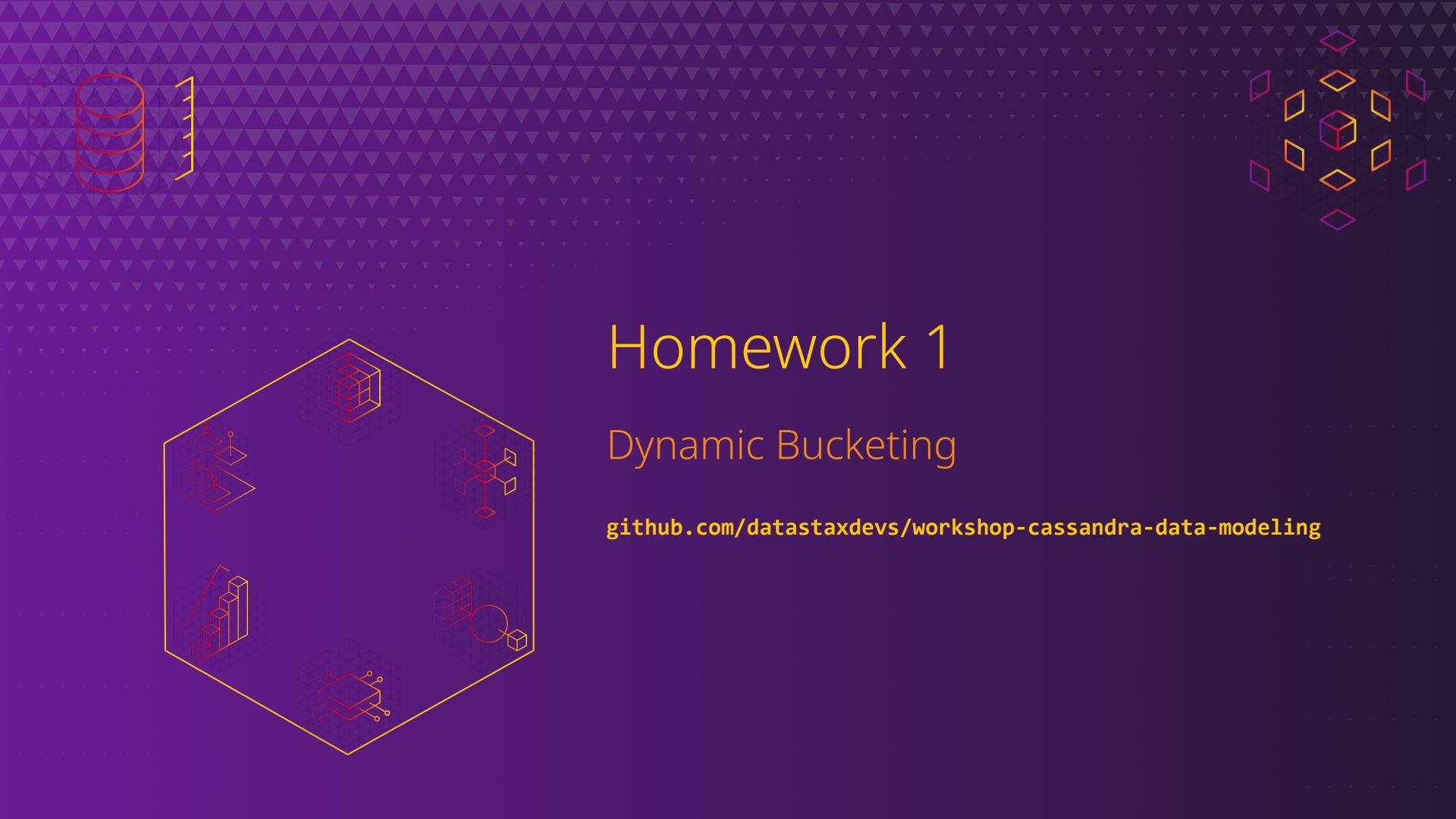
- ❖ Store together what you retrieve together
- ❖ Avoid big partitions
- ❖ **Avoid hot partitions**

```
PRIMARY KEY ((date), sensor, timestamp);
```



```
PRIMARY KEY ((date, sensor), timestamp);
```

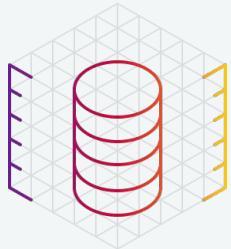




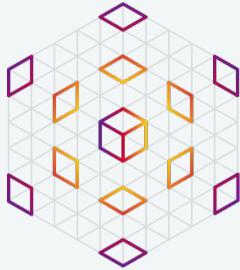
Homework 1

Dynamic Bucketing

github.com/datastaxdevs/workshop-cassandra-data-modeling



» Agenda



01**Data Storage**

Data distribution & organization

02**Key Definition**

Primary, Partition, Clustering

03**CQL Data Types**

Brief review

04**Data Modeling**

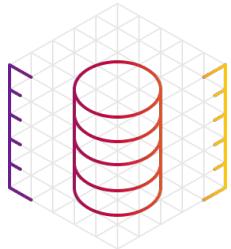
Methodology, notation, tools

05**Optimization Techniques**

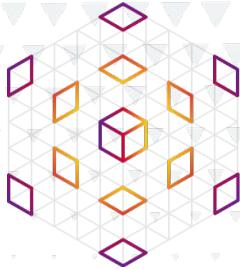
Five important optimizations

06**What's next?**

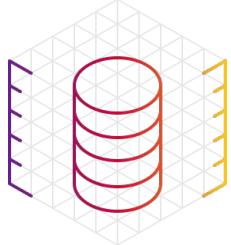
Homework, next sessions



Native 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

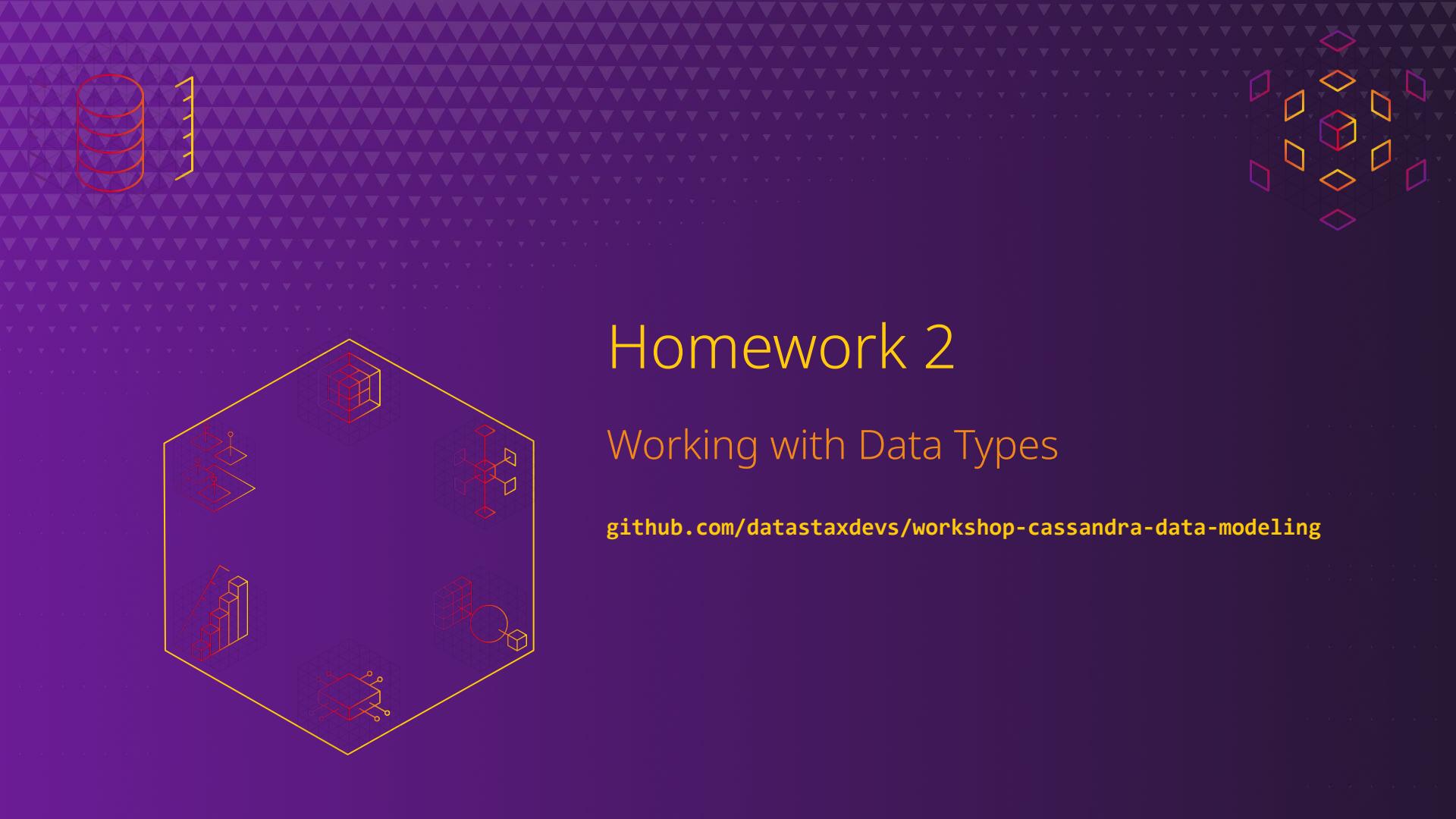


More types

- Collection types
 - `set<int>`
 - `list<text>`
 - `map<int, text>`
- User-Defined Types (UDTs)

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

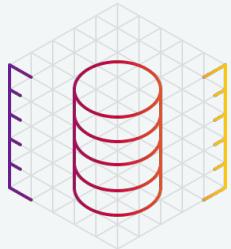
- Tuple types
 - `tuple <uuid, text, decimal>`
- Custom types
 - Your own implementation as a Java class (not recommended)



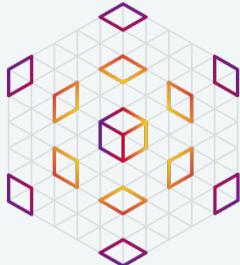
Homework 2

Working with Data Types

github.com/datastaxdevs/workshop-cassandra-data-modeling



» Agenda



01**Data Storage**

Data distribution & organization

02**Key Definition**

Primary, Partition, Clustering

03**CQL Data Types**

Brief review

04**Data Modeling**

Methodology, notation, tools

05**Optimization Techniques**

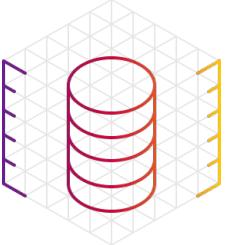
Five important optimizations

06**What's next?**

Homework, next sessions



Normalization vs Denormalization



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.”

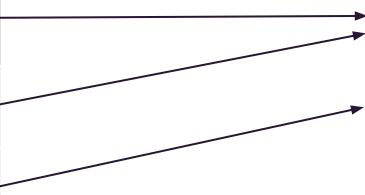
PROS: Simple write, Data Integrity **CONS:** Slow read, Complex Queries

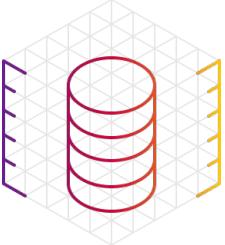
Employees

<u><u>id</u></u>	<u><u>f_name</u></u>	<u><u>l_name</u></u>	<u><u>dept_id</u></u>
1	Edgar	Codd	1
2	Raymond	Boyce	1
3	Alan	Turing	2

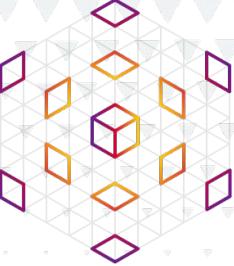
Departments

<u><u>dept_id</u></u>	<u><u>department</u></u>
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

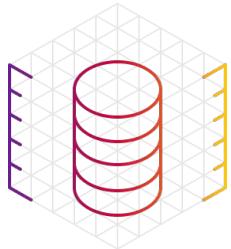
<u>id</u>	<u>f_name</u>	<u>l_name</u>	<u>dept_id</u>	<u>department</u>
1	Edgar	Codd	1	Engineering
2	Raymond	Boyce	1	Engineering
3	Alan	Turing	2	Math

Departments

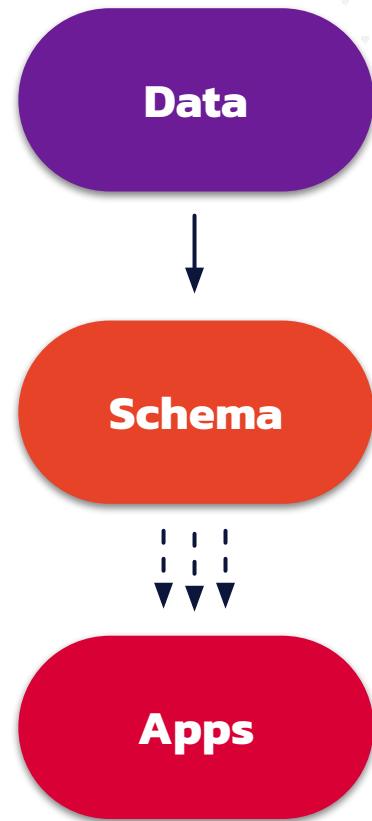
<u>dept_id</u>	<u>department</u>
1	Engineering
2	Math

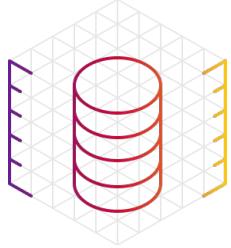


Relational vs NoSQL Schema

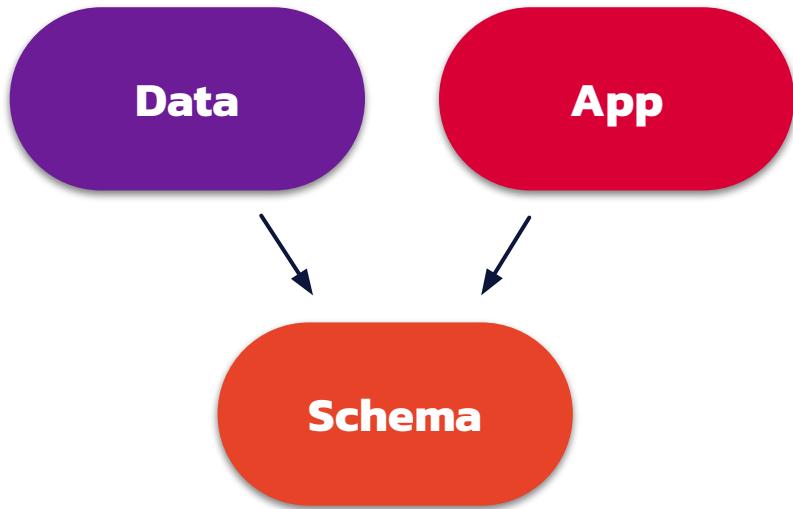
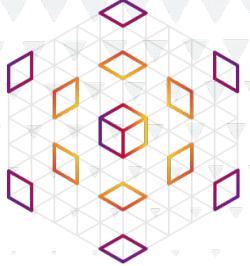


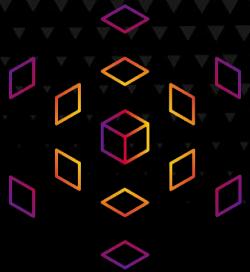
➤ Relational Database = Integration Database



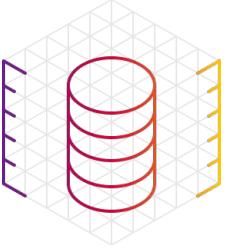


» NoSQL Database = Application Database





Data Modeling Methodology

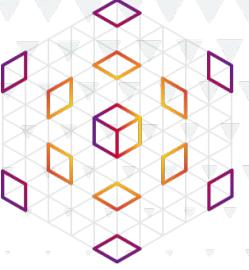


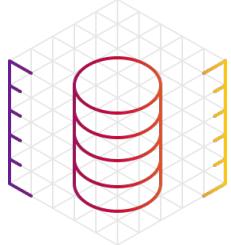
» What is Data Modeling ?

- Collection and analysis of **data requirements**
- Identification of participating entities and relationships
- Identification of data **access patterns**
- A particular way of **organizing** and structuring data
- Design and specification of a **database schema**
- Schema **optimization** and data **indexing** techniques

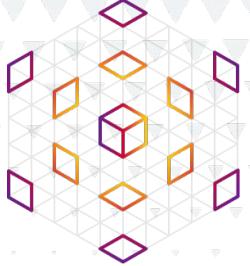


Data Quality: completeness consistency accuracy
Data Access: queryability efficiency scalability





» Cassandra Data Modeling Principles



Modeling principle 1: “**Know your data**”

- Key and cardinality constraints are fundamental to schema design

Modeling principle 2: “**Know your queries**”

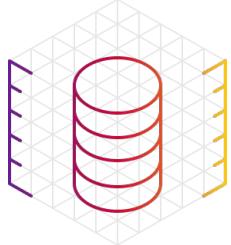
- Queries drive schema design

Modeling principle 3: “**Nest data**”

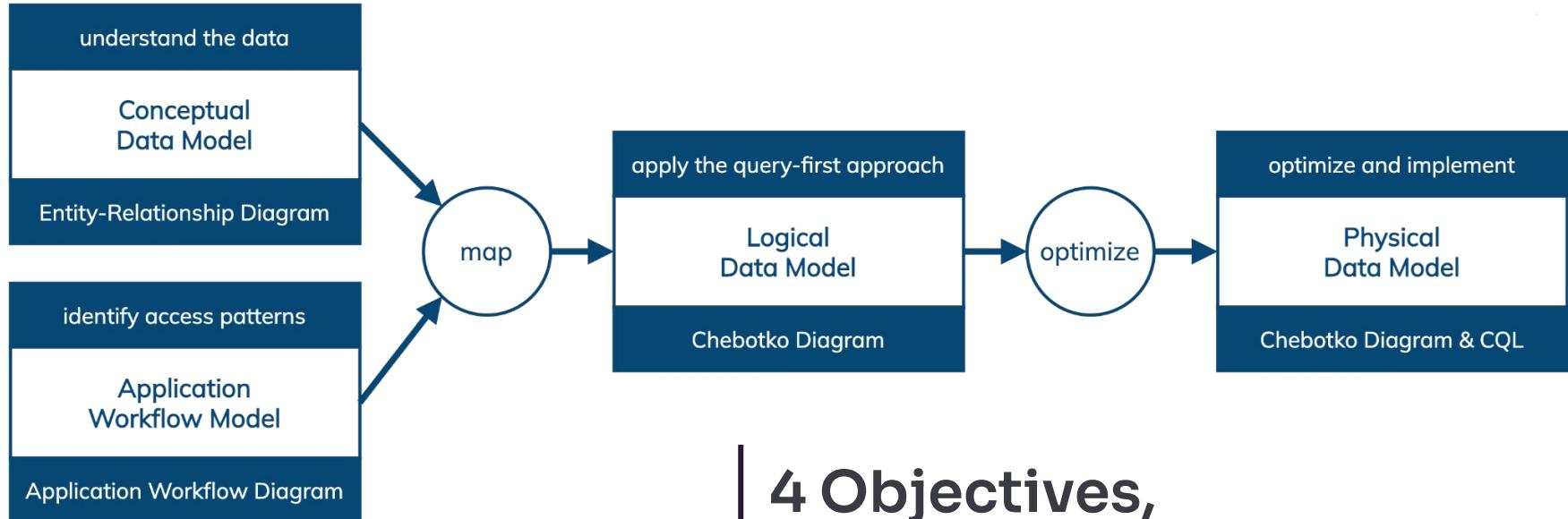
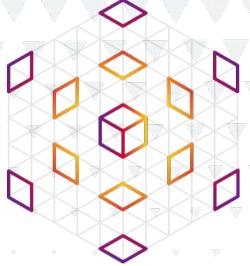
- Data nesting is the main data modeling technique

Modeling principle 4: “**Duplicate data**”

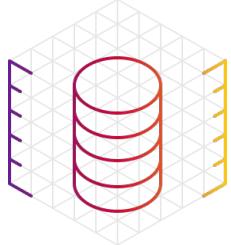
- Better to duplicate than to join



» 4/4/2 Like in soccer (= real football)



**4 Objectives,
4 Models
2 Transitions**



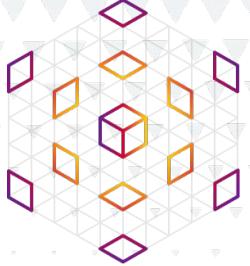
› Data Modeling Methodology: Step I

understand the data

Conceptual
Data Model

Entity-Relationship Diagram

Analyze the Domain



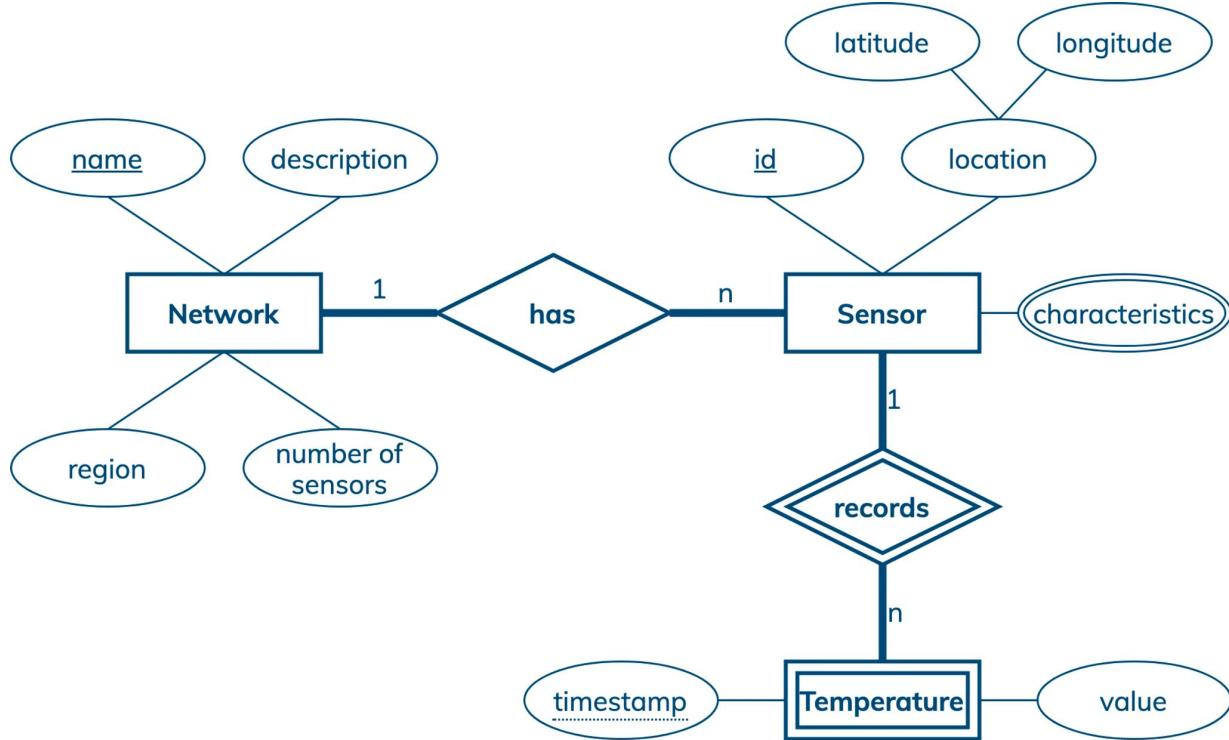
identify access patterns

Application
Workflow Model

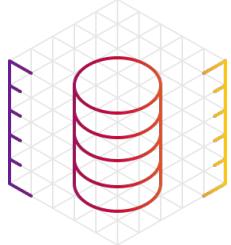
Application Workflow Diagram

Analyze Customer Workflows

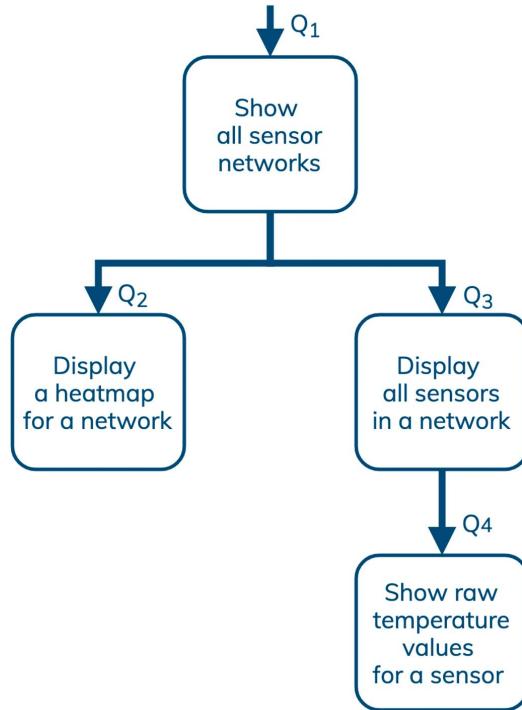
Entity-Relationship Diagram



Entity-Relationship Diagram

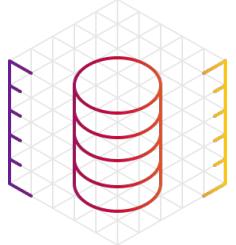


Application Workflow Diagram

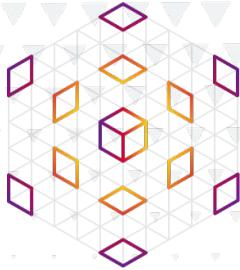
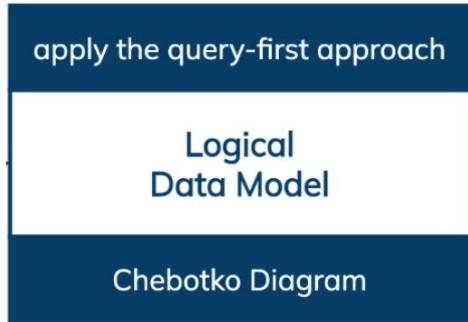


Data access patterns

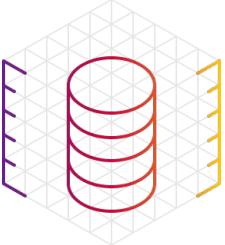
- Q1: Find information about all networks; order by name (asc)
- Q2: Find hourly average temperatures for every sensor in a specified network for a given date range; order by date (desc) and hour (desc)
- Q3: Find information about all sensors in a specified network
- Q4: Find raw measurements for a particular sensor on a specified date; order by timestamp (desc)



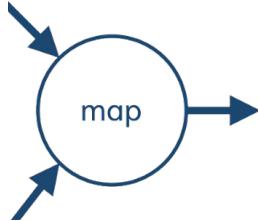
» Data Modeling Methodology: Step II



Design queries and build tables based on the queries



➤ Mapping Rules



Mapping rule 1: “Entities and relationships”

- Entity and relationship types map to tables

Mapping rule 2: “Equality search attributes”

- Equality search attributes map to the beginning columns of a primary key

Mapping rule 3: “Inequality search attributes”

- Inequality search attributes map to clustering columns

Mapping rule 4: “Ordering attributes”

- Ordering attributes map to clustering columns

Mapping rule 5: “Key attributes”

- Key attributes map to primary key columns

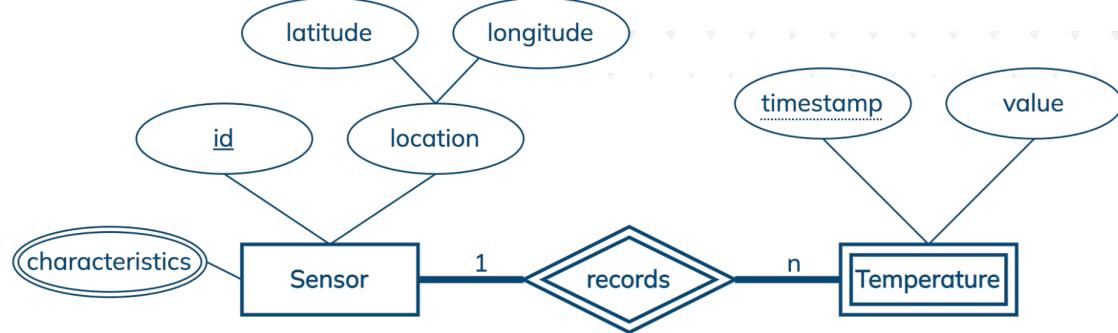
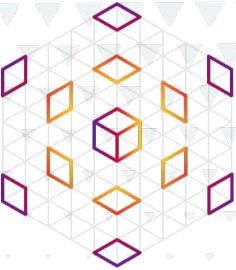
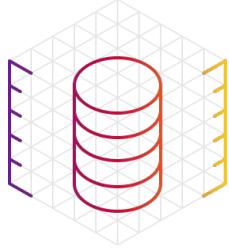
Based on
a conceptual
data model

Based on
a query

Based on
a conceptual
data model



Example: Applying Mapping Rules

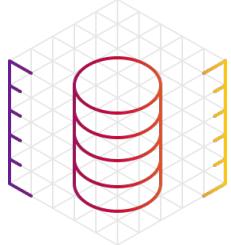


Q5: Find raw measurements for a given location and a date/time range; order by timestamp (desc)

Mapping Rules (MR) applied to the query:

- MR₁**: temps_by_sensor
- MR₂**: temps_by_sensor
- MR₃**: temps_by_sensor
- MR₄**: temps_by_sensor
- MR₅**: temps_by_sensor

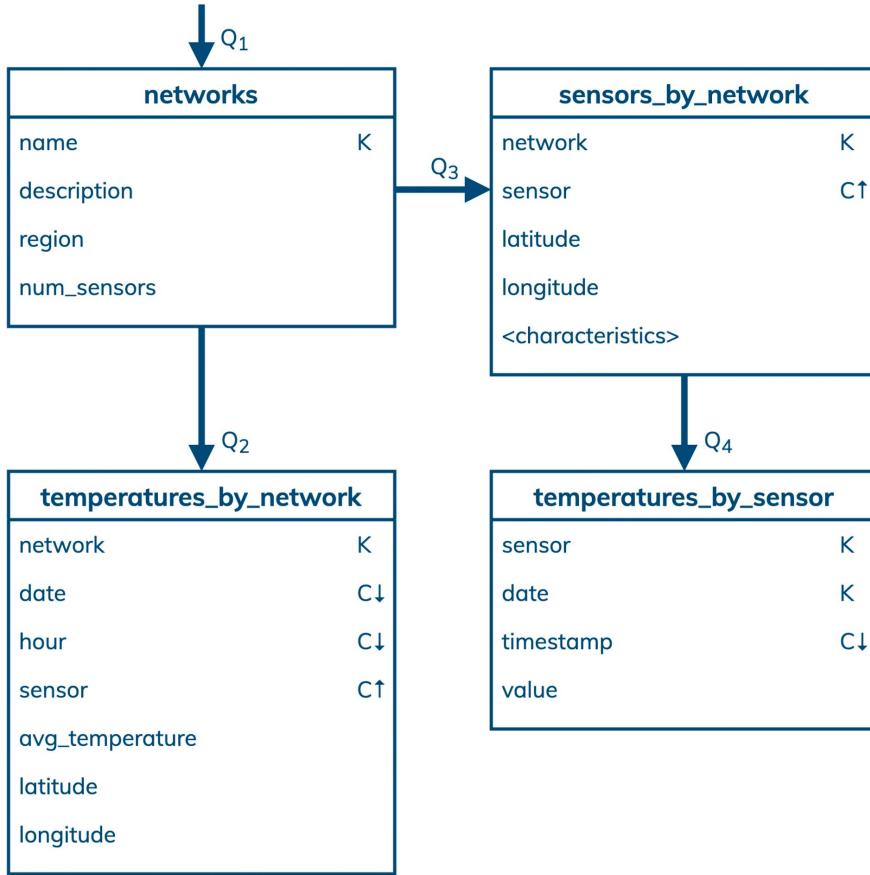
	latitude	longitude	timestamp	value
MR ₁	latitude	longitude	timestamp	value
MR ₂	K	K	C↑	C↓
MR ₃	K	K	C↑	C↑
MR ₄	K	K	C↓	C↑
MR ₅	K	K	C↓	C↑

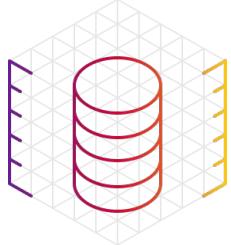


› Chebotko Diagram

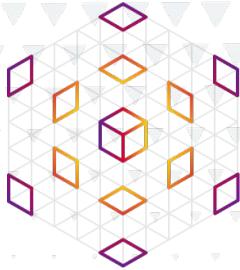
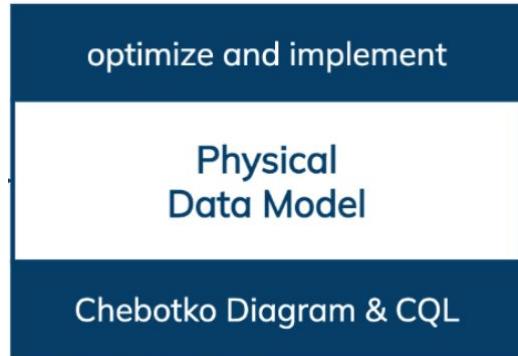
Data access patterns

- Q₁: Find information about all networks; order by name (asc)
- Q₂: Find hourly average temperatures for every sensor in a specified network for a given date range; order by date (desc) and hour (desc)
- Q₃: Find information about all sensors in a specified network
- Q₄: Find raw measurements for a particular sensor on a specified date; order by timestamp (desc)

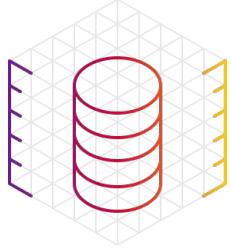




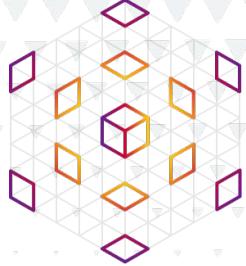
» Data Modeling Methodology: Step III



Optimize and Create

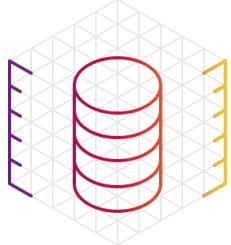


Optimization Techniques

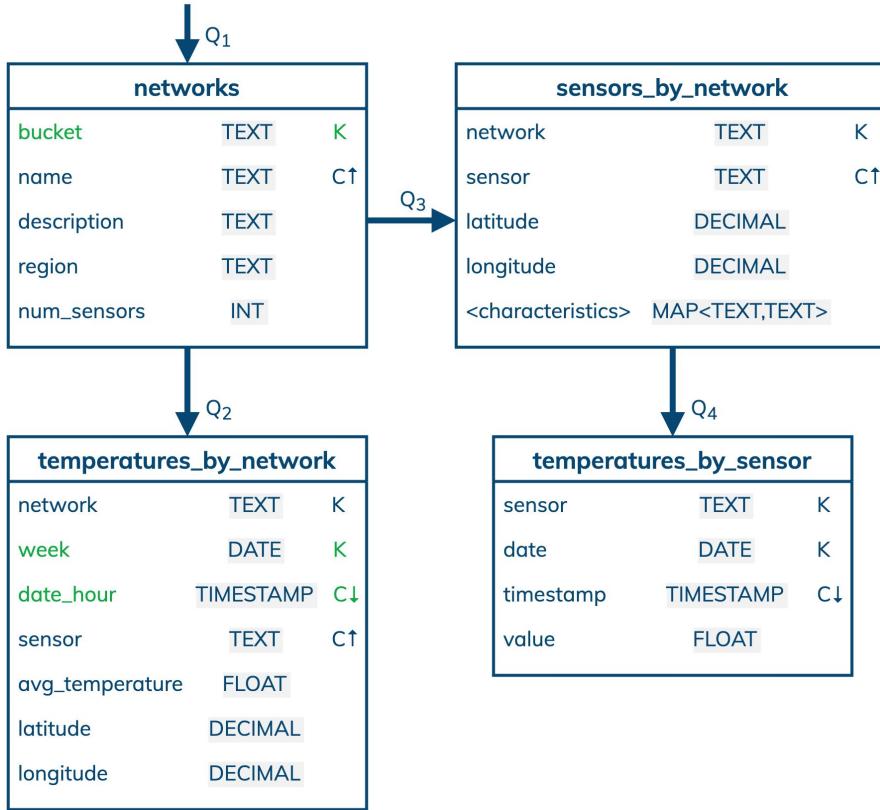


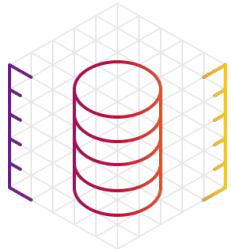
- Partition size limits and splitting large partitions
- Data duplication and batches
- Indexes and materialized views*
- Concurrent data access and lightweight transactions*
- Dealing with tombstones*

* Explained in details at our free DS220 course at academy.datastax.com



› Physical Data Model: Chebotko Diagram





Cassandra Query Language

Q1

```
CREATE TABLE networks (
    bucket TEXT,
    name TEXT,
    description TEXT,
    region TEXT,
    num_sensors INT,
    PRIMARY KEY ((bucket),name)
);
```

Q3

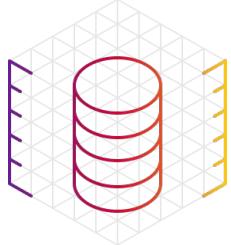
```
CREATE TABLE sensors_by_network (
    network TEXT,
    sensor TEXT,
    latitude DECIMAL,
    longitude DECIMAL,
    characteristics MAP<TEXT,TEXT>,
    PRIMARY KEY ((network),sensor)
);
```

```
CREATE TABLE temperatures_by_network (
    network TEXT,
    week DATE,
    date_hour TIMESTAMP,
    sensor TEXT,
    avg_temperature FLOAT,
    latitude DECIMAL,
    longitude DECIMAL,
    PRIMARY KEY ((network,week),date_hour,sensor)
) WITH CLUSTERING ORDER BY (date_hour DESC, sensor ASC);
```

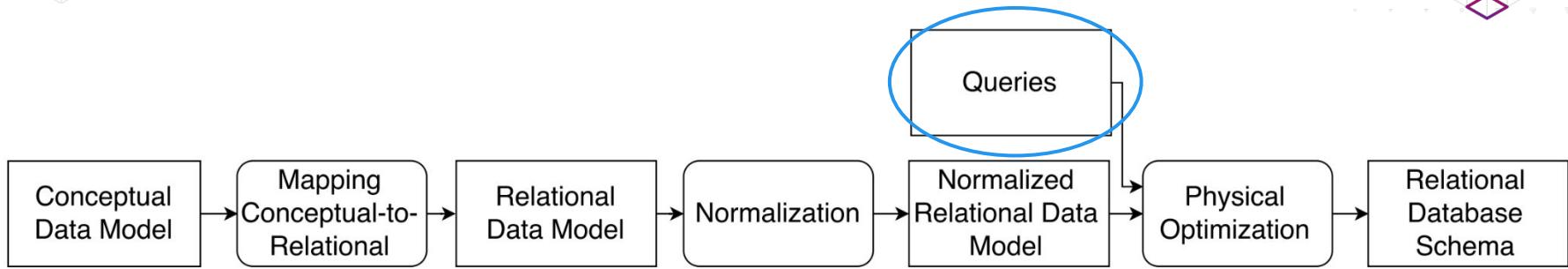
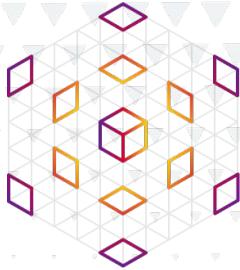
Q2

Q4

```
CREATE TABLE temperatures_by_sensor (
    sensor TEXT,
    date DATE,
    timestamp TIMESTAMP,
    value FLOAT,
    PRIMARY KEY ((sensor,date),timestamp)
) WITH CLUSTERING ORDER BY (timestamp DESC);
```



➤ Relational vs Cassandra Modeling Process



(a) Relational Data Modeling



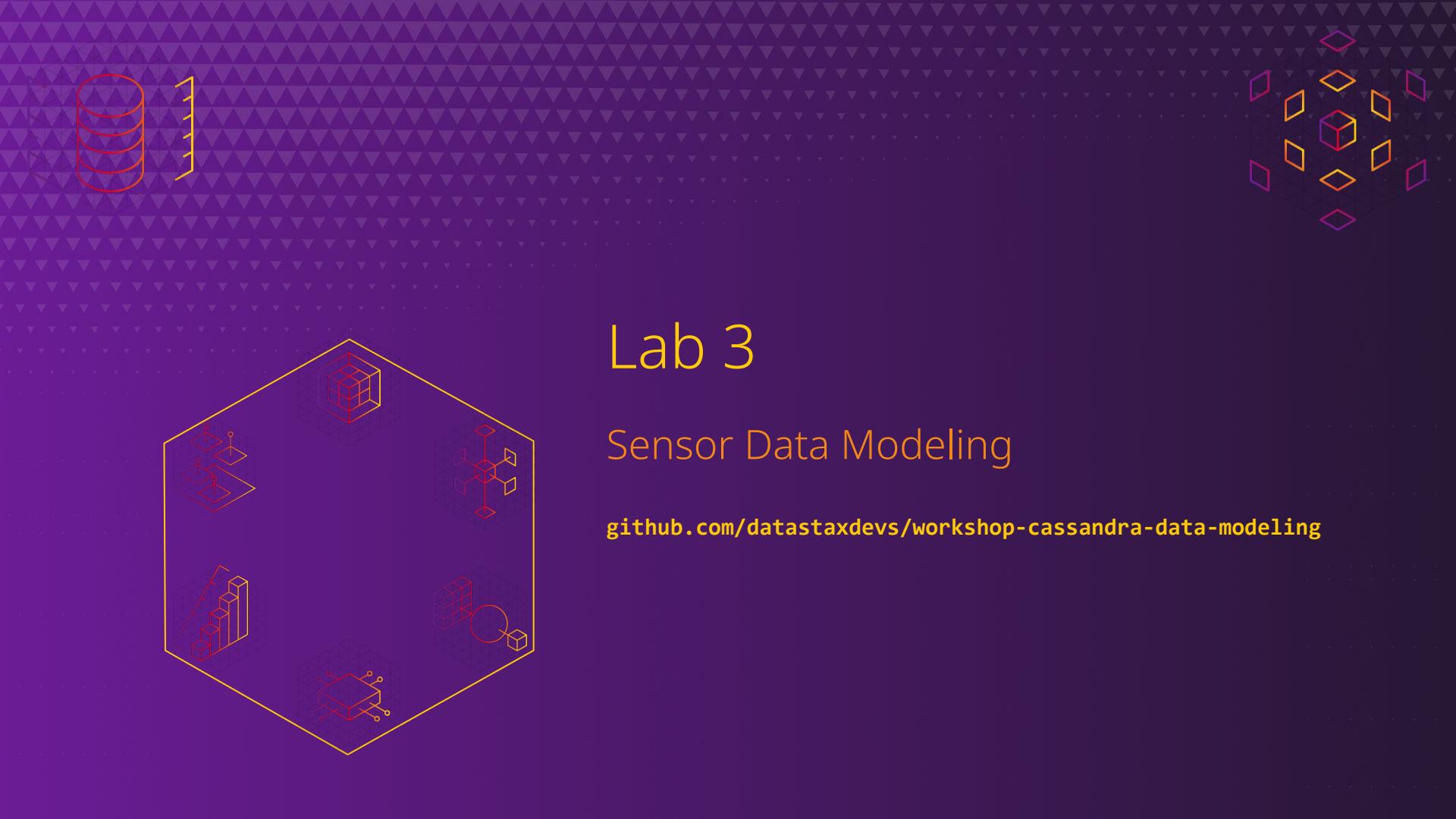
(b) Cassandra Data Modeling



Demo

KDM

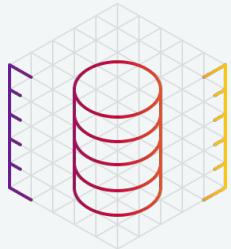
github.com/datastaxdevs/workshop-cassandra-data-modeling



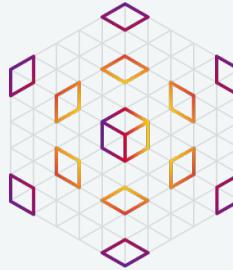
Lab 3

Sensor Data Modeling

github.com/datastaxdevs/workshop-cassandra-data-modeling



» Key Takeaways



Data Modeling Principles

- Know your data
- Know your queries
- Nest data
- Duplicate data

The Most Efficient Access Pattern

A query is satisfied by accessing one partition

Primary Key

Data uniqueness
Data distribution
Data queryability

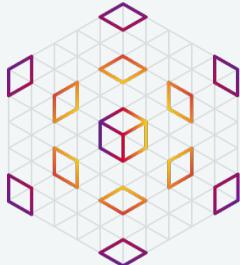
The Cassandra Data Modeling Methodology

It is not more complex than the relational data modeling methodology





» Agenda



01**Data Storage**

Data distribution & organization

02**Key Definition**

Primary, Partition, Clustering

03**CQL Data Types**

Brief review

04**Data Modeling**

Methodology, notation, tools

05**Optimization Techniques**

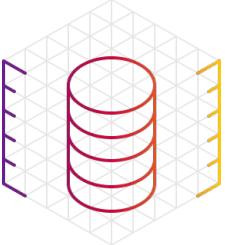
Five important optimizations

06**What's next?**

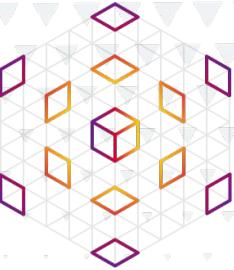
Homework, next sessions



Partition size limits and splitting large partitions

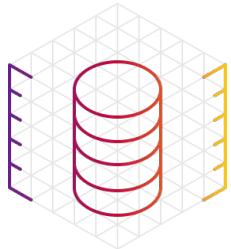


Partition Size Limits

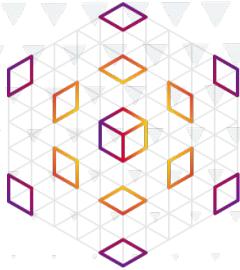


- Theoretical:
 - 2 billion values
 - Node disk size
- Practical:

Cassandra 2	Cassandra 3 and 4
100,000 values	10 x 100,000 values
100 MB	10 x 100 MB



➤ Splitting Large Partitions



- **Solution:** introduce an additional column to a partition key
 - Use an existing or derived column – convenience
 - Use an artificial “bucket” column – more control
- **Cons:** supported access patterns may change

```
PRIMARY KEY ((sensor), timestamp);
```



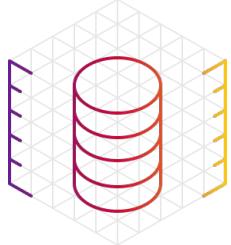
```
PRIMARY KEY ((sensor, date), timestamp);
```





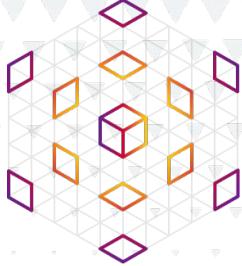
Data duplication and batches

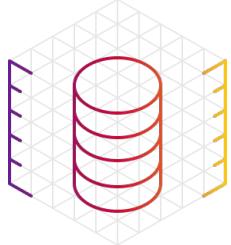




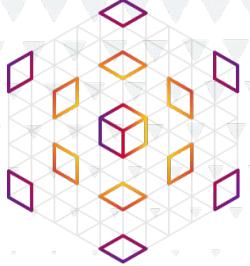
» Data Duplication

- Types of data duplication
 - Duplication across tables
 - Duplication across partitions of the same table
 - Duplication across rows of the same partition
- Duplication factor: constant ✓ vs non-constant ✗





› Keeping Duplicated Data Consistent with **Batches**



- Logged, single-partition batches
 - Efficient and atomic
 - Use whenever possible
- Logged, multi-partition batches
 - Somewhat efficient and pseudo-atomic
 - Use selectively
- Unlogged and counter batches
 - Do not use

BEGIN BATCH

```
INSERT INTO ...;  
INSERT INTO ...;
```

...

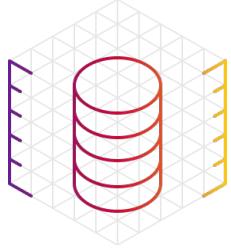
APPLY BATCH;

BEGIN BATCH

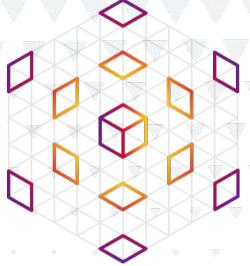
```
UPDATE ...;  
UPDATE ...;
```

...

APPLY BATCH;



› Keeping Duplicated Data Consistent with **MVs**

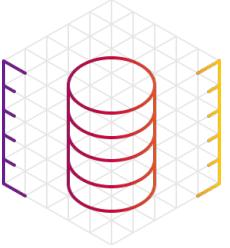


- **Experimental** feature
- Not always applicable
- May become inconsistent
- 10% write penalty to a base table

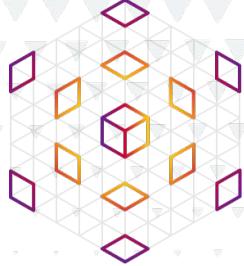
```
CREATE MATERIALIZED VIEW
networks_by_region AS
    SELECT * FROM networks
    WHERE region IS NOT NULL
        AND name IS NOT NULL
PRIMARY KEY ((region), name);
```



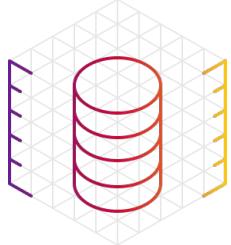
Indexes and materialized views (+ allow filtering)



› Indexing Mechanisms



- Distributed indexes: highly scalable
 - Tables ← **the best solution available**
 - Materialized views (**experimental**)
- Local indexes: **not** highly scalable
 - Secondary indexes (2i)
 - SSTable-attached secondary indexes (SASI)
 - Storage-Attached Indexes (SAI)



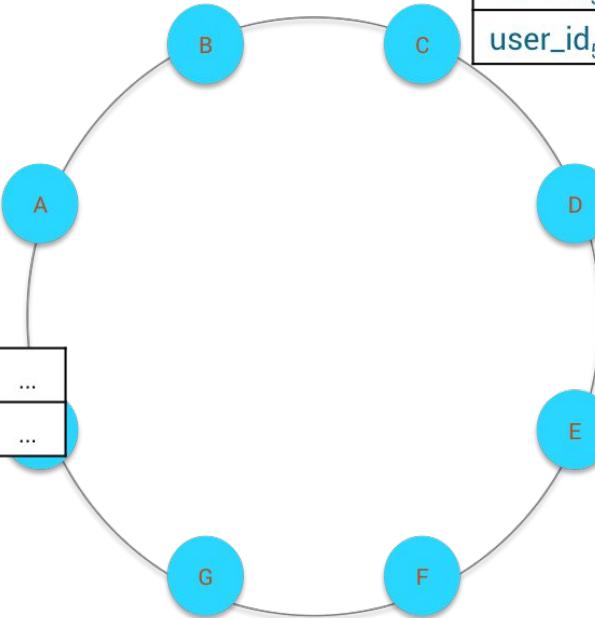
➤ How Secondary Indexes Work

Data on node A

user_id ₁	FR
user_id ₂₃	US
user_id ₇₄	FR

Index on node A

FR	user_id ₁	user_id ₇₄	...
US	user_id ₂₃



Data on node C

user_id ₃	FR
user_id ₃₄	UK
user_id ₅₆	UK

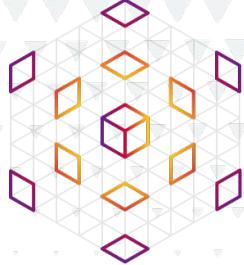
Data on node E

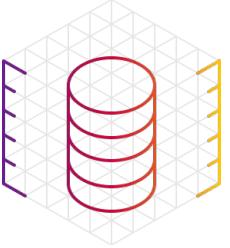
user_id ₂	US
user_id ₄₅	US

Index on node E

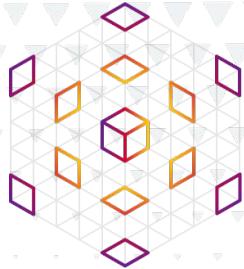
US	user_id ₂	user_id ₄₅	...
----	----------------------	-----------------------	-----

doanduyhai.com/blog/?p=13191



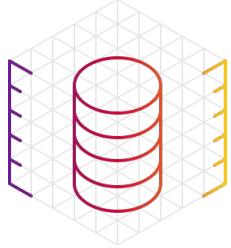


➤ Storage-Attached Indexes (SAI)

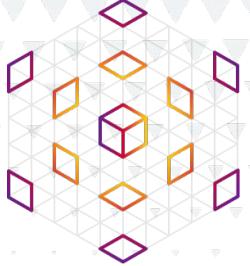


- The best index available
 - Faster writes
 - Less disk space
 - Multiple indexes can be used in one query
 - Reads can be expensive (similar limitations to 2is and SASIs)
- Convenience, not performance

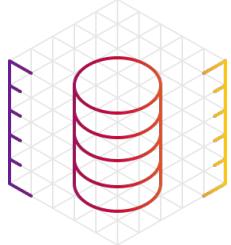
```
CREATE CUSTOM INDEX users_by_name_sai
ON users (name)
USING 'org.apache.cassandra.index.sai.StorageAttachedIndex';
```



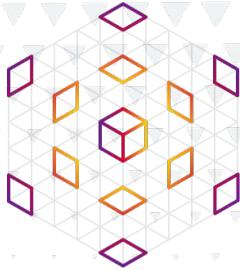
► When to use secondary index (1 of 2)



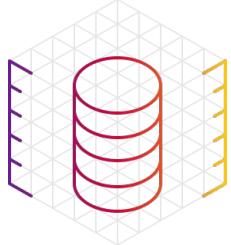
- **Real-time transactional query**: retrieving rows from a large multi-row partition, when both a **partition key** and an indexed column are used in a query.
- **Expensive analytical query**: retrieving a large number of rows from potentially many partitions, when only an indexed **low-cardinality** column is used in a query.



➤ When to use secondary index (2 of 2)



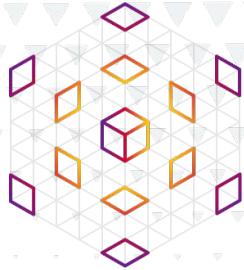
- **Small clusters:** 3-10 nodes with RF=3
- **Infrequent queries:** may not affect the throughput much
- **Queries with LIMIT that can be quickly satisfied:** may not affect the throughput much
- **Convenience over performance:** simpler data modeling while potentially sacrificing throughput and query response time

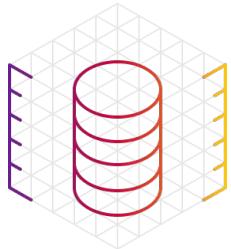


» Materialized Views

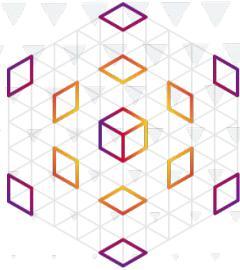
- **Experimental** feature
- Not always applicable
- May become inconsistent
- 10% write penalty to a base table

```
CREATE MATERIALIZED VIEW
networks_by_region AS
    SELECT * FROM networks
    WHERE region IS NOT NULL
        AND name IS NOT NULL
PRIMARY KEY ((region), name);
```

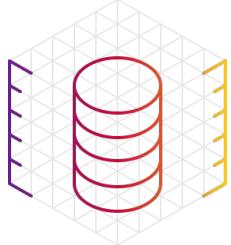




› When to Use a Materialized View



- Queries on higher-cardinality columns
 - Similar advantages as those of regular tables
 - Convenience of automatic view maintenance
 - Reads are as fast as for regular tables
- Important limitations
 - Restrictions on how PRIMARY KEY is constructed
 - 10% slower writes to the base table
 - Base-view inconsistencies
 - Use LOCAL_QUORUM and higher for base table writes
 - **Disabled in Astra DB, experimental**

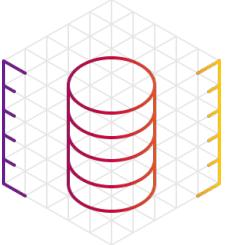


Allow Filtering

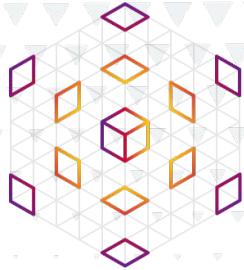
- Generally, using ALLOW FILTERING is not recommended
- But **scanning may be cheaper than indexing** in some cases
 - Scanning within a small partition
 - Scanning within a large partition based on a low-cardinality column
(most rows from the partition will be returned)
 - Scanning a table based on a low-cardinality column for analytical purposes
(most rows from the table will be returned)



Concurrent data access and lightweight transactions

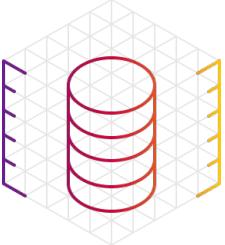


➤ Race conditions

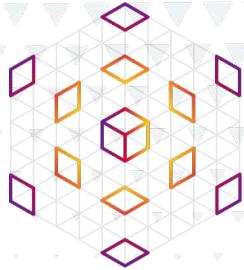


- Multiple transactions reading and writing the same piece of data concurrently, possibly leaving the data in a state **inconsistent with reality**
- Examples: registering a new username, counting votes, updating account balance, updating an order status, updating the highest bid

User 1:	read(10)	10<20	write(20)	-----> time
User 2:		read(10)	10<15	write(15)



› Linearizable Consistency



- Linearizable consistency ensures that concurrent transactions produce the same result as if they execute in a sequence, one after another
- Lightweight transactions (**LWTs**) in Cassandra and Astra DB

```
User 1: read(10) 10<20 write(20)
```

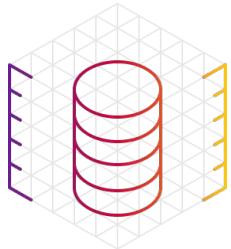
-----> time

```
User 2:           read(20) 20>15
```

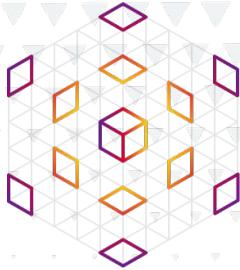
```
User 1:           read(15) 15<20 write(20)
```

-----> time

```
User 2: read(10) 10<15 write(15)
```



› Lightweight transactions (**LWTs**)

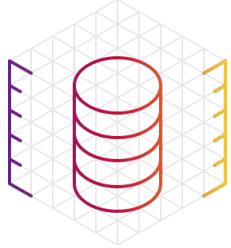


- Guarantee linearizable consistency
- Use with **race conditions and low data contention**
- Require 4 coordinator-replica round trips
- May become a bottleneck with high data contention

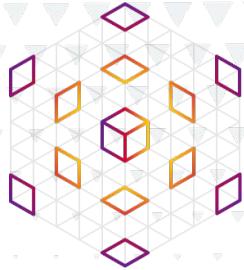
```
INSERT INTO ... VALUES ...
IF NOT EXISTS;

UPDATE ... SET ... WHERE ...
IF EXISTS | IF predicate [ AND ... ];

DELETE ... FROM ... WHERE ...
IF EXISTS | IF predicate [ AND ... ];
```



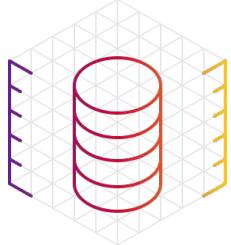
➤ Dealing with High Data Contention



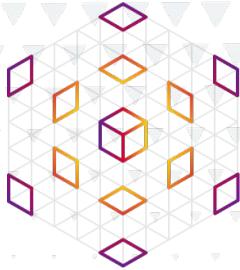
- Change a data model to eliminate race conditions
 - Isolate computation by isolating data
 - Record each transaction separately and aggregate data later
- Use a queue to control data contention
 - Astra Streaming



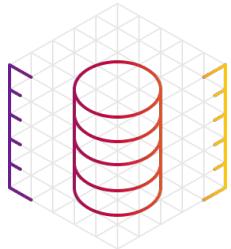
Tombstones



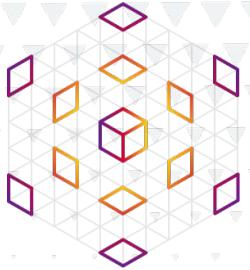
› Tombstones



- Written markers for deleted data
 - DELETEs
 - Data with TTL
 - UPDATEs and INSERTs with explicit NULL values
 - “Overwritten” collections
 - Multiple types: cell, row, range, partition, TTL tombstones for cells
 - Cleaned up during compaction after `gc_grace_seconds`



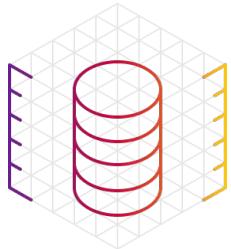
› Tombstone Issue 1: Large Partitions



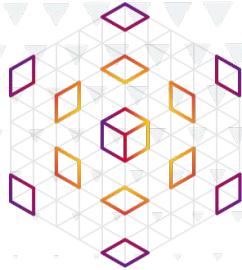
- Tombstone = additional data to store and read
- Query performance degrades, heap memory pressure increases
- `tombstone_warn_threshold`
 - Warning when more than 1,000 tombstones are scanned by a query
- `tombstone_failure_threshold`
 - Aborted query when more than 100,000 tombstones are scanned



- Consider **LCS** (leveled-compaction strategy + tuning) to ease read load
- Decrease `gc_grace_seconds` (default is 864000 or 10 days)
 - Deleted data and tombstones can be purged during compaction after `gc_grace_seconds`
 - Adjust repair frequency accordingly! (see next slide)



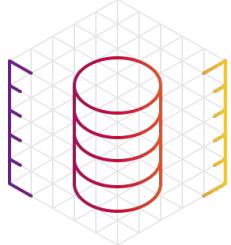
› Tombstone Issue 2: Zombie or Resurrected Data



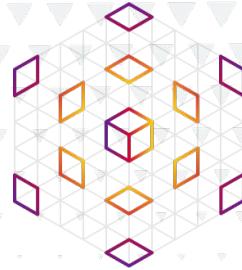
1. A replica node is unresponsive and receives no tombstone
2. Other replica nodes receive tombstones
3. The tombstones get purged after `gc_grace_seconds` and compaction
4. The unresponsive replica comes back online
and *resurrects* data that was previously marked as deleted



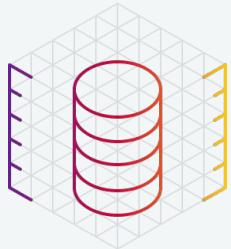
- Run repairs within the `gc_grace_seconds` and on a regular basis
- `nodetool repair` manually when needed
- Do not let a node rejoin the cluster after the `gc_grace_seconds`
 - but if you do, start it in `rebuild` mode after deleting `data/`



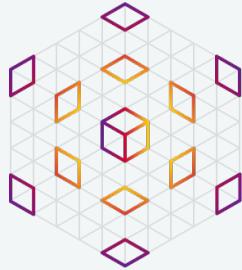
➤ Other Tombstones Considerations



- Design data models that avoid frequent deletes
- Use the most coarse-grained delete possible:
(truncate) > partition > range > row > cell
 - Truncating/dropping a table is better than deleting each partition in the table
 - Deleting a partition is better than deleting each row in the partition
 - Deleting a row is better than deleting each value in the row



» Key Takeaways



Partition Size

100K and 100MB

Bucketing

Data duplication and batches

Avoid non-constant duplication factors

Single-partition batches are efficient

Indexing

Materialized views are scalable but have other issues

Secondary indexes are good for searching within large partitions

LWTs

Linearizable consistency

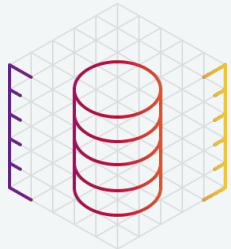
Race conditions and low data contention

Design data models that eliminate race conditions

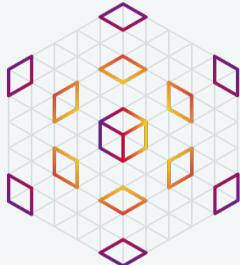
Tombstones

Design data models that avoid frequent deletions

Compact and repair to purge tombstones and avoid resurrected data



» Agenda



01**Data Storage**

Data distribution & organization

02**Key Definition**

Primary, Partition, Clustering

03**CQL Data Types**

Brief review

04**Data Modeling**

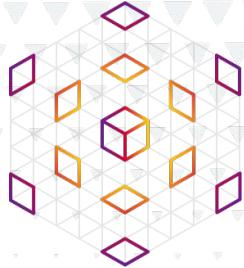
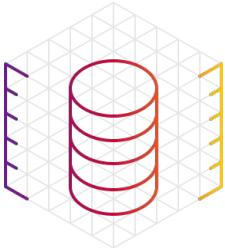
Methodology, notation, tools

05**Optimization Techniques**

Five important optimizations

06**What's next?**

Homework, next sessions



Section 1 of 3

Cassandra Data Modelling Homework



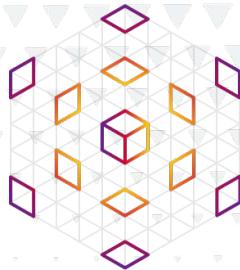
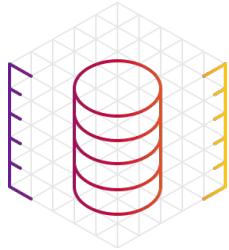
Welcome and thank you! Here you can submit your homework for the datastax developers "Data Modelling with Cassandra" workshop. In case of any questions please contact organizers at <https://dtsx.io/aleks> or just send an email to aleksandr.volochnev@datastax.com

- Workshop materials: <https://github.com/datastaxdevs/workshop-intro-to-cassandra>
- Discord chat: <https://dtsx.io/discord>

Email *

Valid email

This form is collecting emails. [Change settings](#)



datastax.com/learn/data-modeling-by-example

LEARNING SERIES

Data Modeling by Example

This growing collection of data modeling examples from various domains will help you get started on your own Apache Cassandra™ database schema design. Each example applies our Cassandra Data Modeling Methodology to produce and visualize four important artifacts: conceptual data model, application workflow model, logical data model and physical data model. Last but not least, the hands-on portion of each example makes it easy to implement a data model in Cassandra, express data access patterns as CQL queries and run the queries against our sample dataset. Get a functioning proof of concept for your use case in minutes!

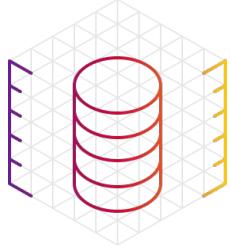
GET STARTED



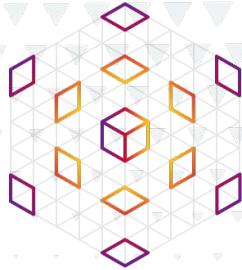
Learning Series Topics

DURATION
APPROXIMATELY 3H

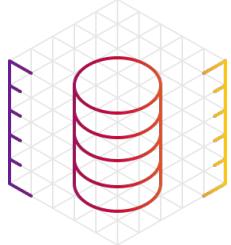
- | | |
|---------------------------------------|-----------------------------------|
| 01 Sensor Data Modeling | 05 Time Series Data Modeling |
| 02 Messaging Data Modeling | 06 Shopping Cart Data Modeling |
| 03 Digital Library Data Modeling | 07 Order Management Data Modeling |
| 04 Investment Portfolio Data Modeling | |



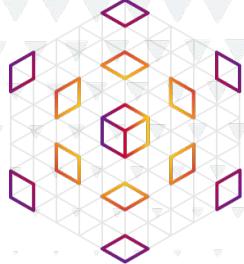
➤ Further readings



- Data Modeling in Cassandra and DataStax Astra DB
medium.com/building-the-open-data-stack/data-modeling-in-cassandra-and-datastax-astra-db-3f89b9c133c9
- Five Data Models for IoT
devm.io/iot/apache-cassandra-iot-174970



Stay in touch!

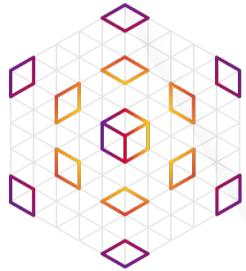
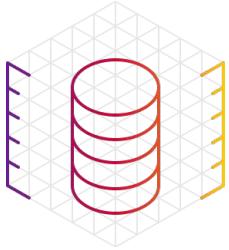


Discord: dtsx.io/discord

Academy: academy.datastax.com

Workshops: datastax.com/workshops

YouTube: [@DataStax Developers](#)



Thank You