Cassandra Day

2022

50 Pitfalls of Cassandra Developers

https://dtsx.io/cday-pitfall

Sponsored by DataStax

Cédrick Lunven, Director of Developer Relations

Astra SDK

cassandra

Java

FF4J

{ REST }

GRPC

- Trainer
- Public Speaker
- Developers Support
- Developer Applications
- Developer Tooling

- Creator of ff4j (ff4j.org)
- Maintainer for 8 years+

- Happy developer for 14 years
- Spring Petclinic Reactive & Starters
- Implementing APIs for 8 years

clunven

clunven

clun

Agenda

01

Data Modeling
The good, the bad, the ugly

02

Shape your requests up !
It is not “just CQL”

03

Cassandra Graveyard
Tombstones and Zombies

04

Developers Horror Museum
Session, Object Mapping, Frameworks

05

Administration and Operation
Scale like a boss

Agenda

01

Data Modeling
The good, the bad, the ugly

02

Shape your requests up !
It is not “just CQL”

03

Cassandra Graveyard
Tombstones and Zombies

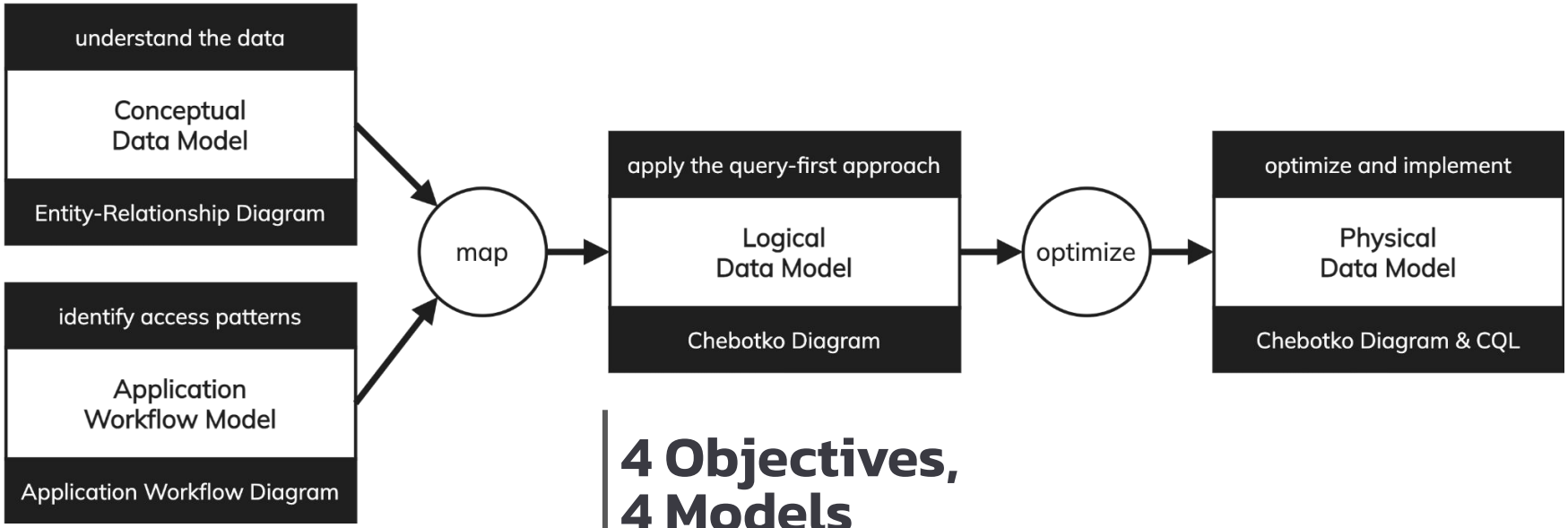
04

Developers Horror Museum
Session, Object Mapping, Frameworks

05

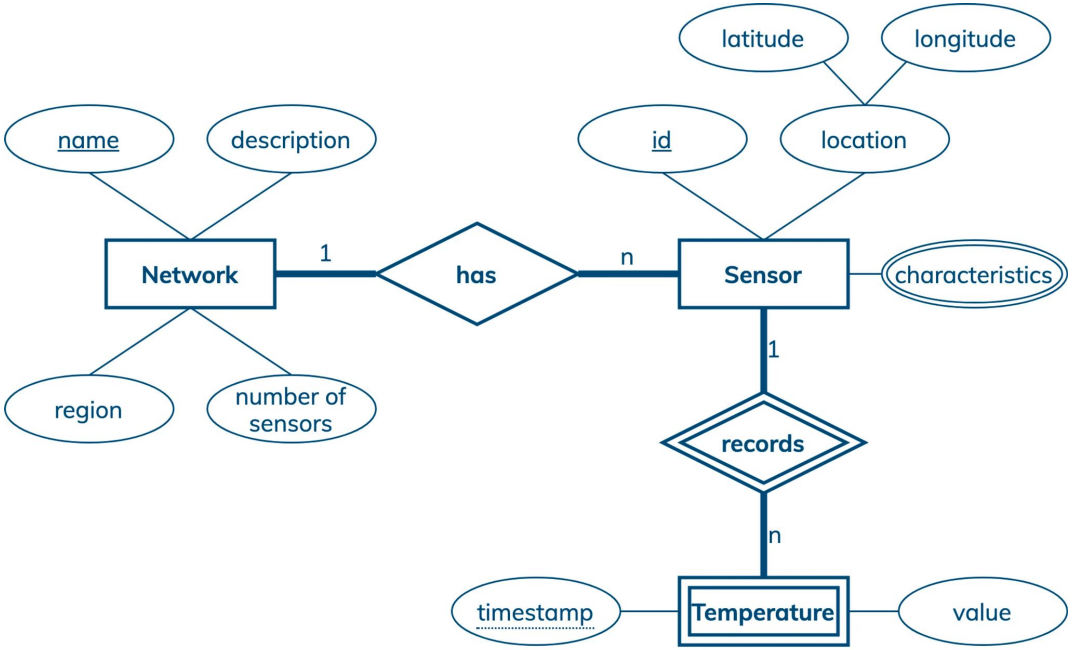
Administration and Operation
Scale like a boss

Data Modeling Methodology

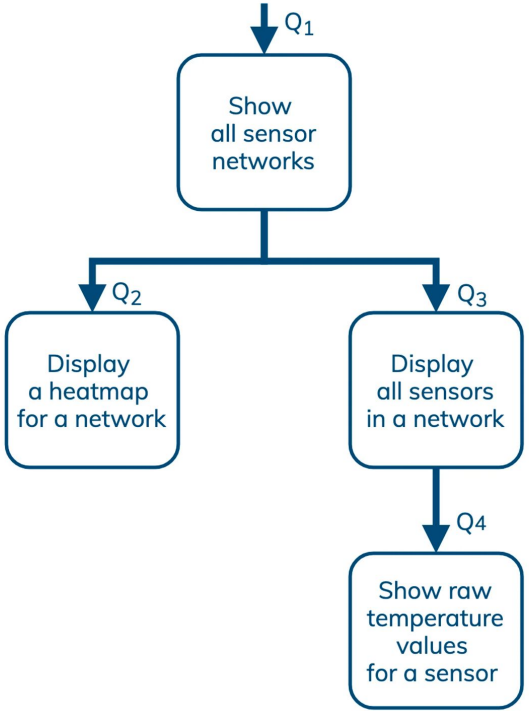


4 Objectives,
4 Models
2 Transitions

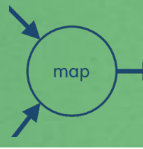
Data Modeling in action



Entity-Relationship Diagram



Data Modeling Methodology



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

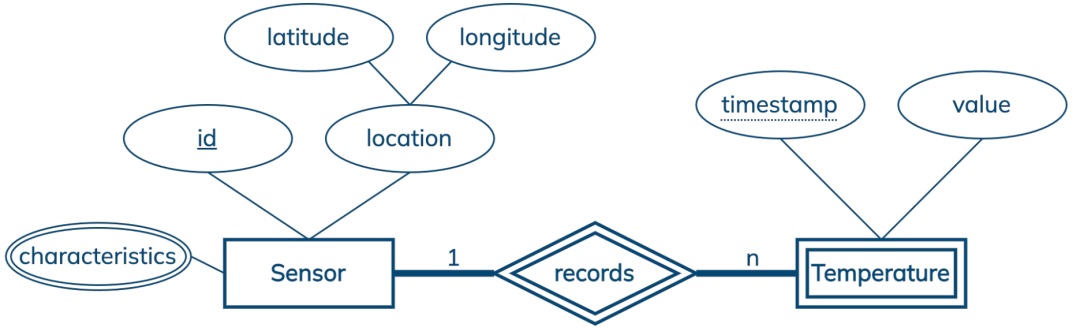
Mapping rule 5: "Key attributes"
- Key attributes map to primary key columns

Based on a conceptual data model

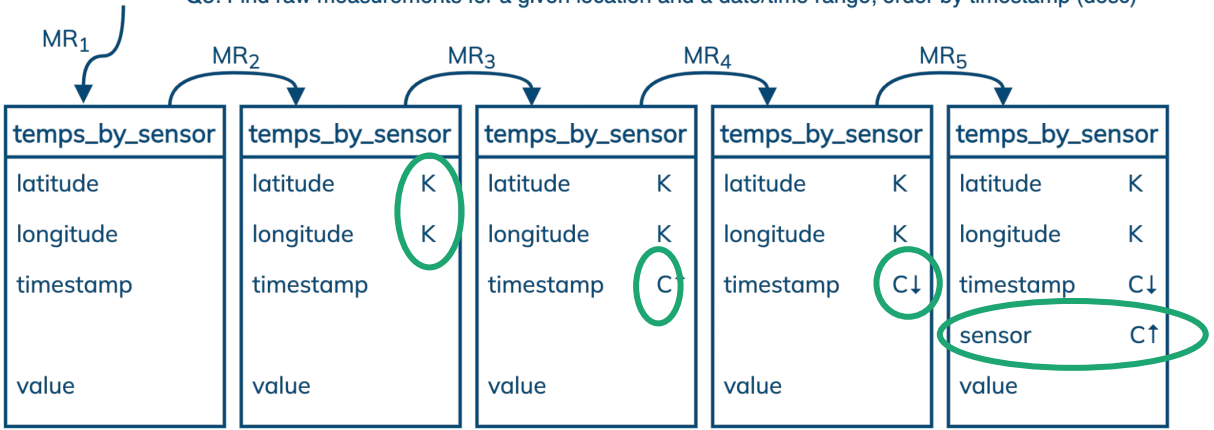
Based on a query

Based on a conceptual data model

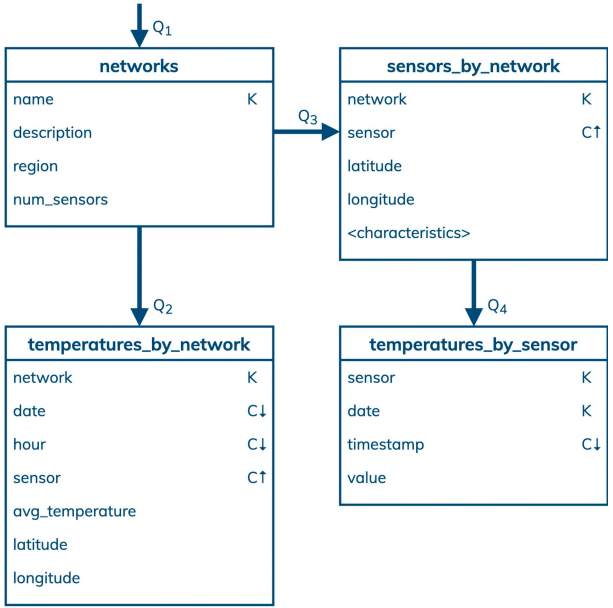
Data Modeling in action



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



Some data modeling (High level) pitfalls



❌ DO NOT

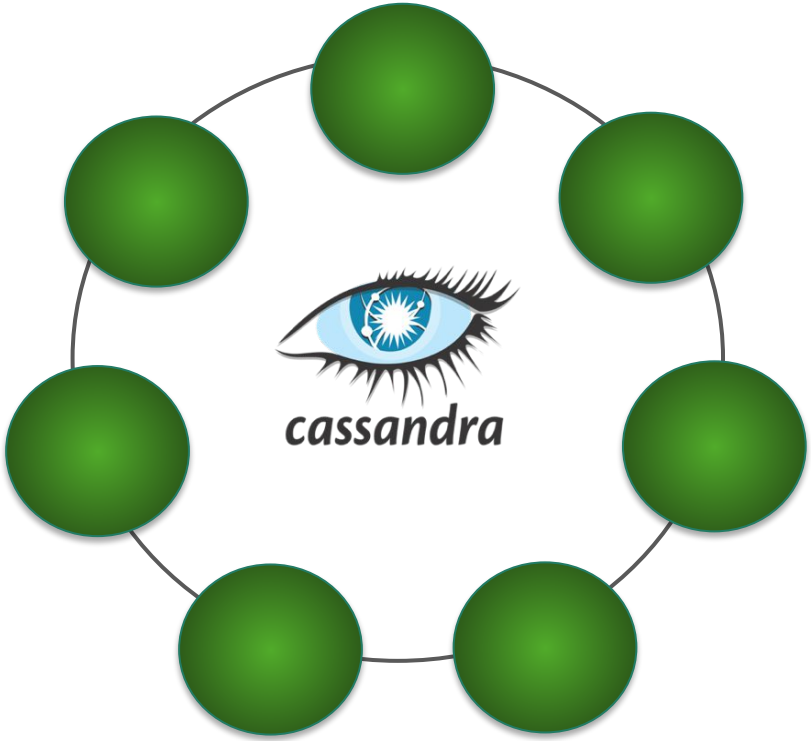
- Use relational DB methodology (3NF…) but follow Cassandra methodology
- Reuse same tables for different where clauses

✅ DOs

- Know your requests and schema => data model before code
- 1 table = 1 query (most of the time)
- Duplicate the data
- Secondary indexes as last resort
- Materialized views are experimental

More to come on types, collections…

Partition Sizing : Balanced small partitions

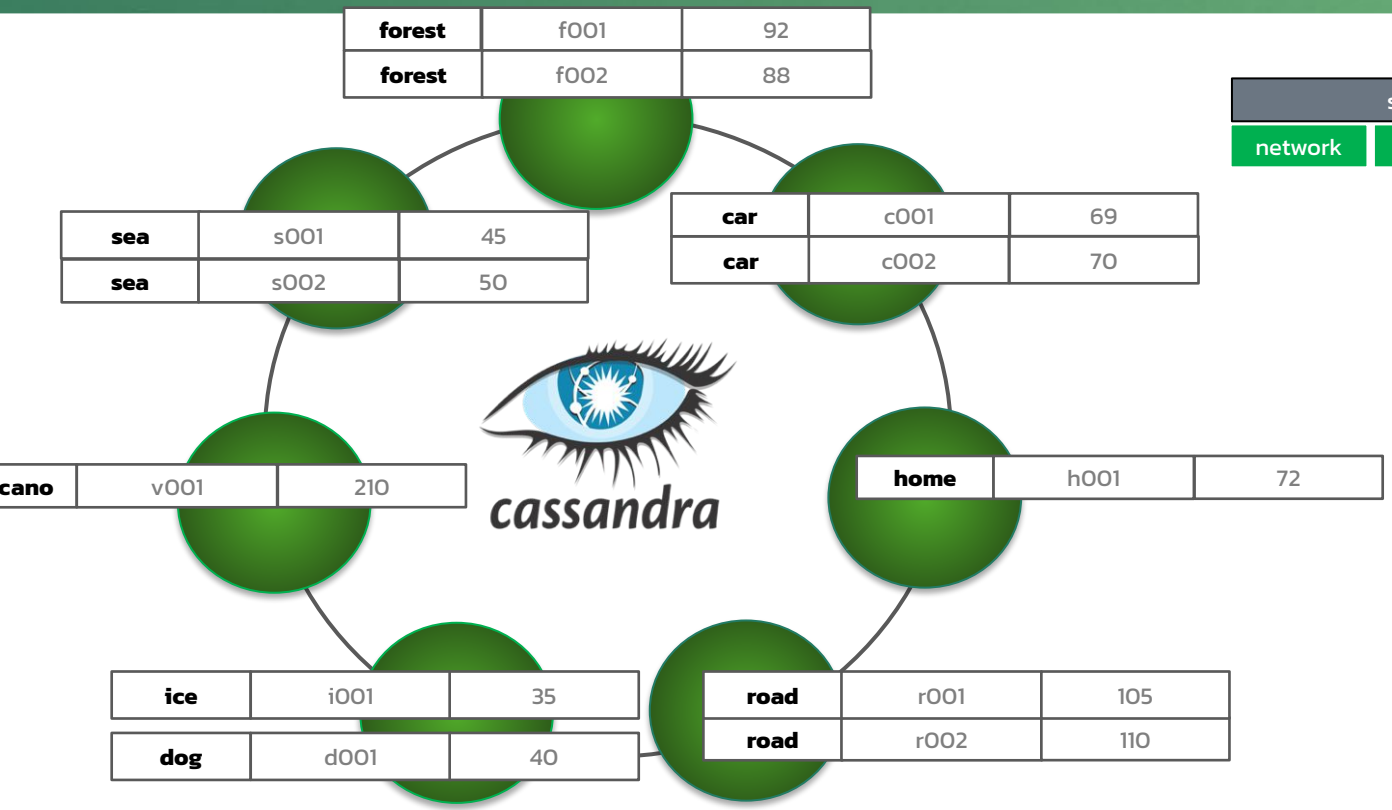


sensors_by_network		
network	sensor	temperature
forest	f001	92
forest	f002	88
volcano	v001	210
sea	s001	45
sea	s002	50
home	h001	72
car	c001	69
car	c002	70
dog	d001	40
road	r001	105
road	r002	110
ice	i001	35

Partition Key

Primary Key

Partition Sizing : Balanced small partitions



sensors_by_network		
network	sensor	temperature

Partition Sizing : Balanced small partitions

```
CREATE TABLE temperatures_by_XXXX (
  sensor      TEXT,
  date        DATE,
  timestamp   TIMESTAMP,
  value       FLOAT,
  PRIMARY KEY ????
)...
```

```
PRIMARY KEY ((sensor));
```
Not Unique ✗

```
PRIMARY KEY ((sensor), value, timestamp);
```
Not sorted ✗

```
PRIMARY KEY ((sensor), timestamp);
```
Big partition ✗

```
PRIMARY KEY ((date), sensor, timestamp);
```
Hot partition ✗

BUCKETING

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

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

Partition Sizing : Balanced small partitions

📘 Sizing limits:

- Soft: 100k Rows, 100MB and 10MB/cell (slightly bigger with C_4)
- Hard: 2 billions cells

✅ Calculators Tools:

$$\sum_i sizeOf(c_{k_i}) + \sum_j sizeOf(c_{s_j}) + N_r \times \left(\sum_k sizeOf(c_{r_k}) + \sum_l sizeOf(c_{c_l}) \right) + sizeOf(t_{avg}) \times N_v$$

- Cql Calculator:

https://github.com/johnnywidth/cql-calculator

- Cassandra Data Modeler

http://www.sestevez.com/sestevez/CassandraDataModeler/

C_k: pk columns C_r: regular columns

C_s: static columns C_c: clustering columns

N_r: Number of rows

N_v: number of values = Nr * (Nc-Mpk-Ns)+ Ns

T_{avg}: avg cell metadata (~8 bytes)

✅ Bench

List collection type

Ordered list of values, can contains duplicates, can access data with index.

❌ Limitations

- Server side race condition when simultaneously adding and removing elements.
- Setting and removing an element by position incur an internal read-before-write.
- Prepend or append operations are non-idempotent.
- There is an additional overhead to store the index

✅ Solutions

- If order is not important use Set
- if list will be “large” (100+) use dedicated tables
- if very small “(<10)” could be clustering columns

```
CREATE TABLE IF NOT EXISTS table_with_list (
  uid      uuid,
  items    list<text>,
  PRIMARY KEY (uid)
);

INSERT INTO table_with_list(uid,items)
VALUES (c7133017-6409-4d7a-9479-07a5c1e79306,
['a', 'b', 'c']);

UPDATE table_with_list SET items = items + ['f']
WHERE uid = c7133017-6409-4d7a-9479-07a5c1e79306;
```

Set and Map collection types

📘 What

- Same structures as java, unordered, ensure unicity
- Conflict-free replicated types = same value on each server
- Frozen: content serialized as a single value
- Non-frozen (default): save as a separate values

❌ Limitations (non-Frozen)

- Overhead to store metadata
- Reading one value still returned the full collection
- Tombstones are created (more soon)
- Performance degradation over time (updates in different SSTABLES)

✅ Solutions

- Use frozen when data is immutable

```
CREATE TABLE IF NOT EXISTS table_with_set (
  uid      uuid,
  animals  set<text>,
  PRIMARY KEY (uid)
);

CREATE TABLE IF NOT EXISTS table_with_map (
  uid         text,
  dictionary  map<text, text>,
  PRIMARY KEY (uid)
);
```

User Defined Types (UDT)

📘 What

- Structures created by user to store organized data
- Hold a schema

❌ Limitations (non-Frozen)

- Schema evolutions: you cannot delete UDT columns
- Mutation size will increase (too many elements or too much nested UDT). Hitting `max_mutation_size` = failed operation.

✅ Solutions

- Use frozen UDT when it is possible
- When using non-UDT limit the number of columns
- Falling back to text based column containing JSON releases pressure

```
CREATE TYPE IF NOT EXISTS udt_address (
  street text,
  city text,
  state text,
);

CREATE TABLE IF NOT EXISTS table_with_udt (
  uid      text,
  address   udt_address,
  PRIMARY KEY (uid)
);
```

Counters

📘 What:

- 64-bit signed integer, imprecise values such as likes, views
- Two operations only: increment, decrement
- First op assumes value is zero
- NOT LIKE ORACLE SEQUENCES, NOT AUTO-INCREMENT

❌ Limitations

- Cannot be part of primary key
- Cannot be mixed with other types in table
- Cannot be inserted or updated with a value
- Updates are not idempotent
- Writes are slower (extra local read ar replica level)

✅ Solutions

- Tables containing multiple counters should be distributed (counter name part of PK)

```
CREATE TABLE IF NOT EXISTS table_with_counters (
  handle        text,
  following     counter,
  followers     counter,
  notifications counter,
  PRIMARY KEY (handle)
);

UPDATE table_with_counters
SET followers = followers + 1
WHERE  handle = 'clunven';
```

Agenda

01

Data Modeling
The good, the bad, the ugly

02

Shape your requests up !
It is not “just CQL”

03

Cassandra Graveyard
Tombstones and Zombies

04

Developers Horror Museum
Session, Object Mapping, Frameworks

05

Administration and Operation
Scale like a boss

Cassandra Query Language

❌ Don't

- AVOID SELECT *, provides column names, adding columns gets you more data than expected.
- AVOID SELECT COUNT(*), will likely timeout, will spread across all nodes, use **DsBulk**
- AVOID (LARGE) IN(...) statements across partitions requests, inefficient and will hit different nodes, N+1 select pattern
- AVOID ALLOW ~~DUMBERING~~ FILTERING, full scan cluster 😨

✅ Dos

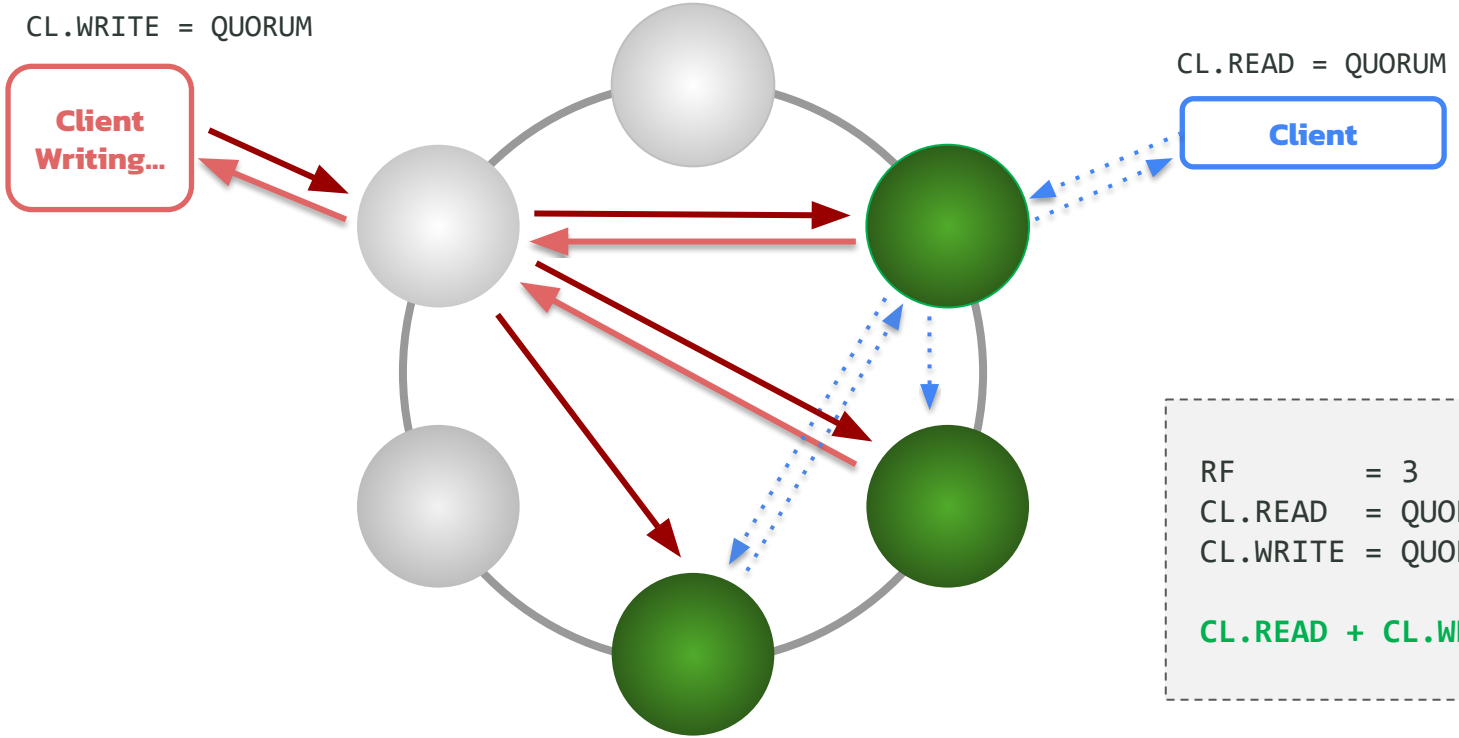
- Prepare your statements (Parse once, run many times)
  - Saves network trips for result set metadata.
  - Client-side type validation.
  - Statements binding on partition keys compute their own cluster routing.
- Know your Cql Types
  - Reduce space usage on disk
  - https://cassandra.apache.org/doc/latest/cassandra/cql/types.html
- Use Metadata : TTL, TIMESTAMP

```
PreparedStatement ps = session
  .prepare("
SELECT id FROM sensors_by_network
WHERE network = ?");

BoundStatement psb = ps.bind(network);

ResultSet rs = session.execute(psb);
```

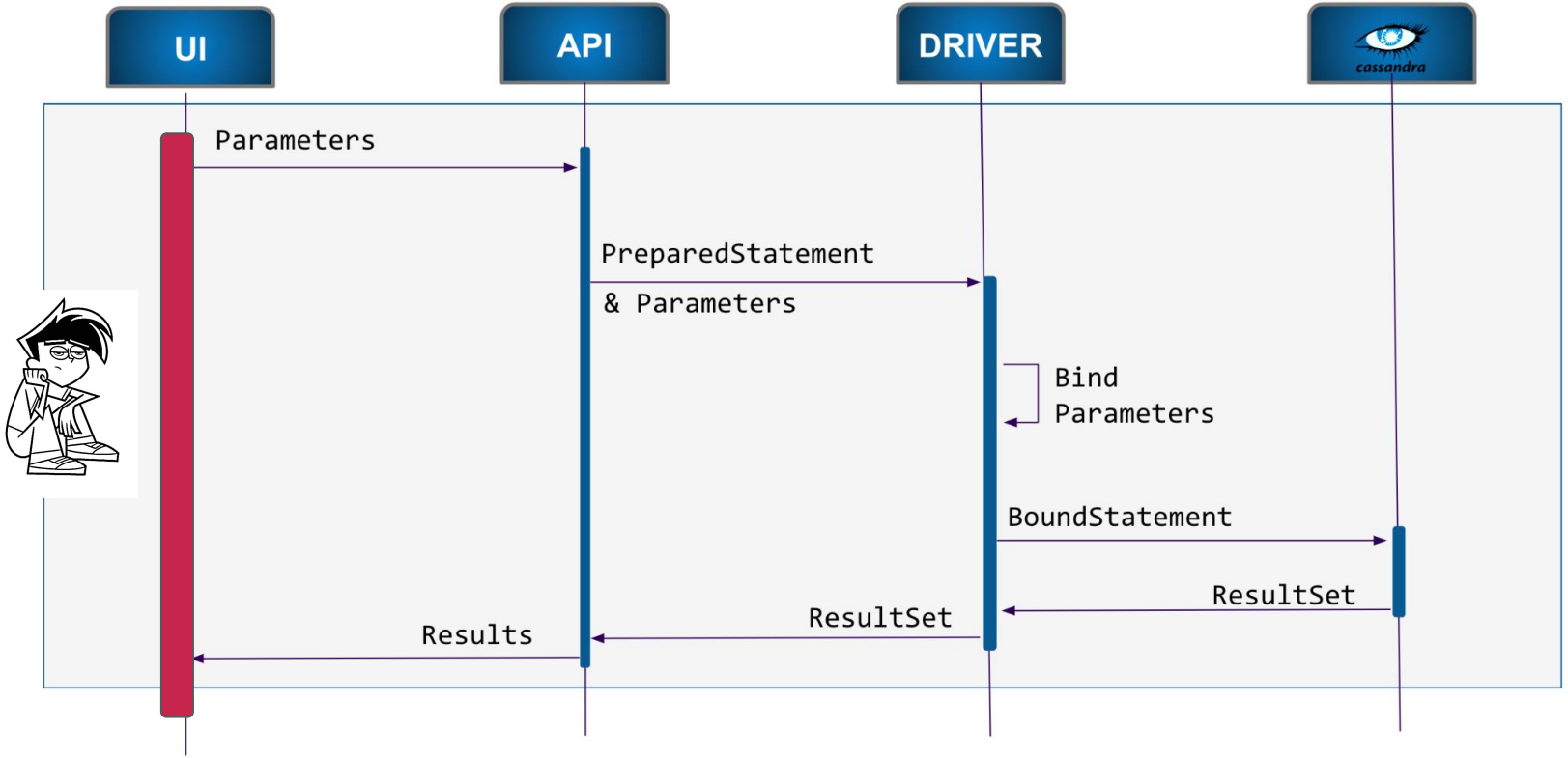
Enforce Immediate Consistency



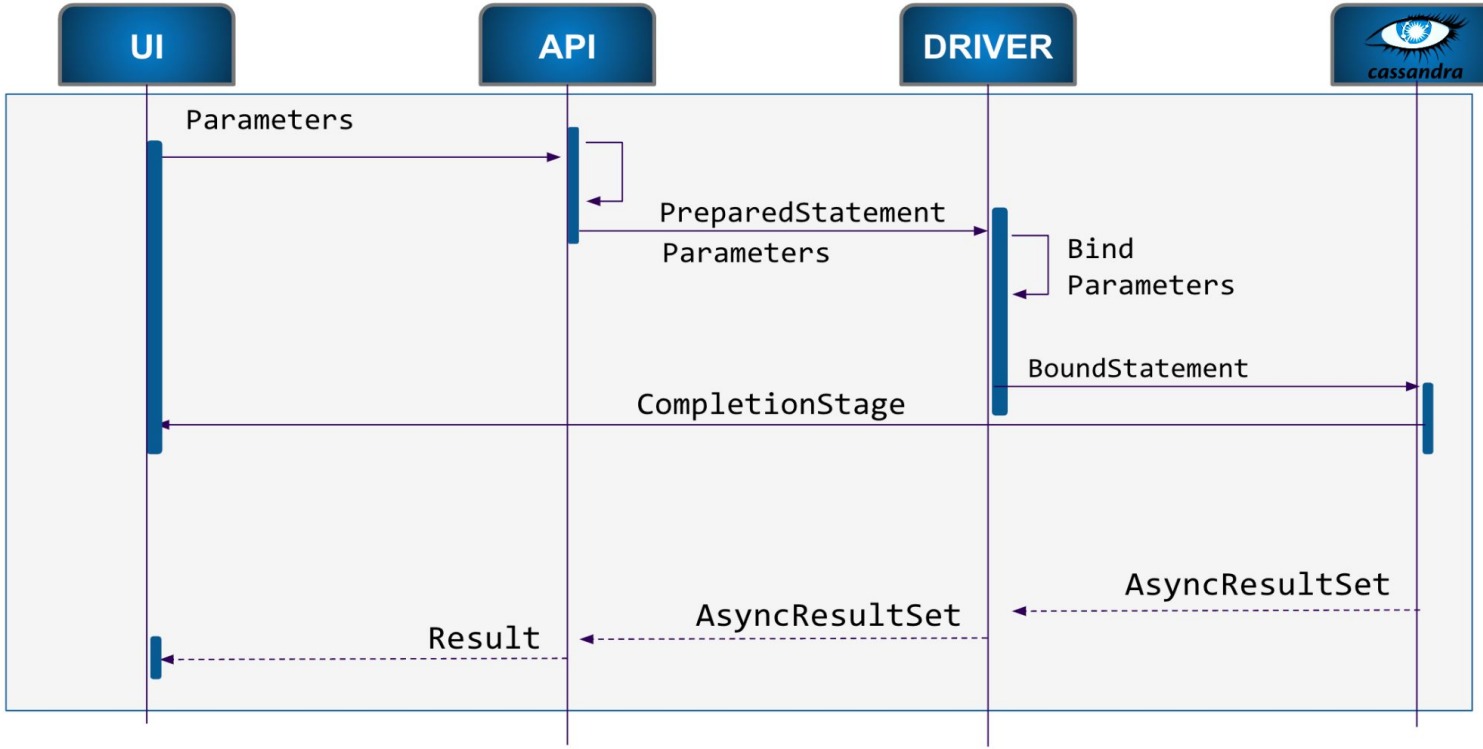

```
RF        = 3
CL.READ   = QUORUM = RF/2 + 1 = 2
CL.WRITE  = QUORUM = RF/2 + 1 = 2

CL.READ + CL.WRITE > RF --> 4 > 3
```

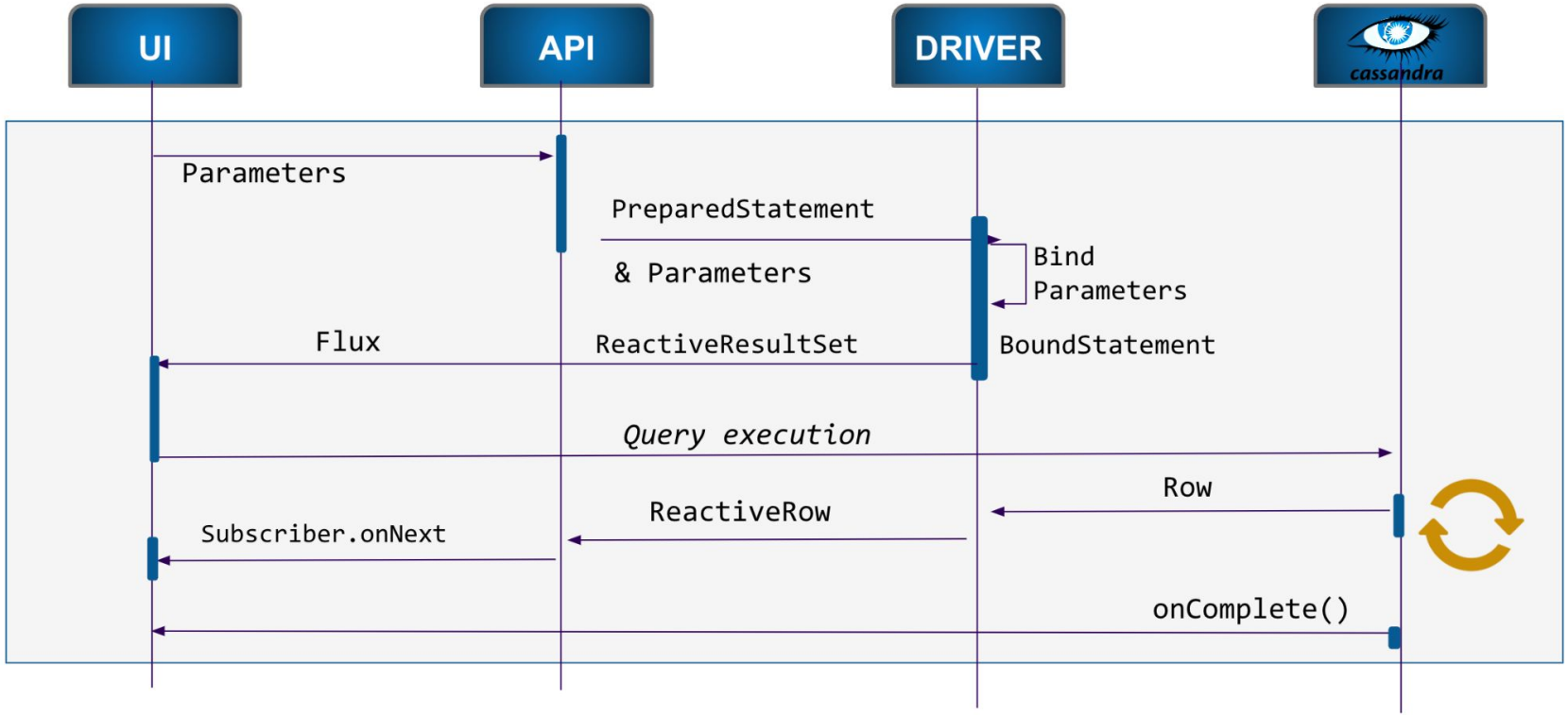
Synchronous Requests



Asynchronous Requests



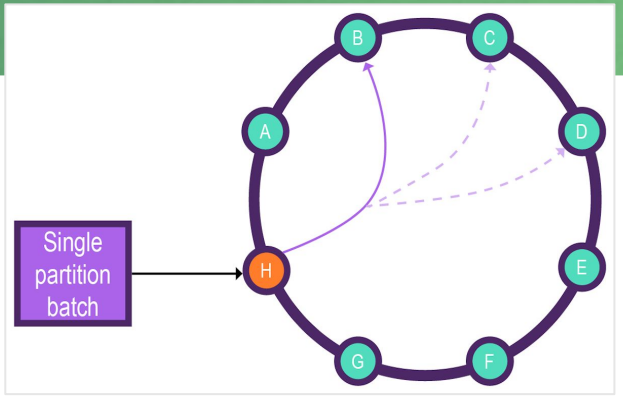
Reactive Requests



Understanding Batches

What

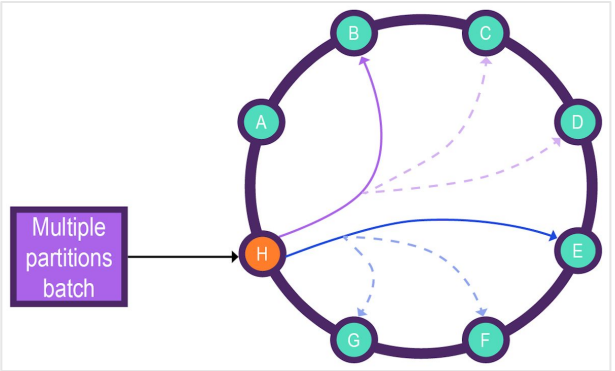
- Execute multiple modification statements (insert, update, delete) simultaneously
- Coordinator node for batch, then each statements executed at the same time
- Retries until it works or times out



Use Cases

- ✅ 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;
```



Lightweight Transaction (LWT)

📘 What

Linearizable consistency ensures that concurrent transactions produce the same result as if they executed in a sequence, one after another.

- Guarantee linearizable consistency
- Require 4 coordinator–replica round trips

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

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

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

❌ Don’t

- May become a bottleneck with high data contention

✅ Dos

- Use with race conditions and low data contention

Agenda

01

Data Modeling
The good, the bad, the ugly

02

Shape your requests up !
It is not “just CQL”

03

Cassandra Graveyard
Tombstones and Zombies

04

Developers Horror Museum
Session, Object Mapping, Frameworks

05

Administration and Operation
Scale like a boss

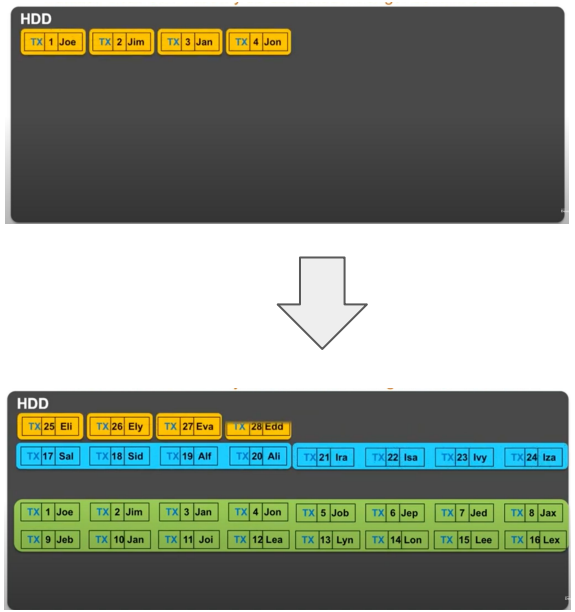
Understanding compaction strategy

📘 About Compaction Strategy

- Process to merge SSTABLES in bigger ones
- Defined per table

✅ Dos

- **SizeTiered Compaction (STCS)** – (Default)
  - Triggers when multiple SSTables of a similar size are present.
  - Insert-heavy and general use cases
- **Leveled Compaction (LCS) –**
  - groups SSTables into levels, each of which has a fixed size limit which is 10 times larger than the previous level.
  - Read-Heavy workload (no overlap at same level), I/O intensive
- **TimeWindow Compaction (TWCS) –**
  - creates time windowed buckets of SSTables that are compacted with each other using the Size Tiered Compaction Strategy.
  - Immutable Data



Tombstones

```
[ { "partition" : {
    "key" : [ "CA102" ], "position" : 0,
    "deletion_info" : {
      "marked_deleted" : "2020-07-03T23:11:58.785298Z",
      "local_delete_time" : "2020-07-03T23:11:58Z"
    } },"rows" : [ ] },

  { "partition" : {
    "key" : [ "CA101" ],
    "position" : 20  },
    "rows" : [
  { "type" : "row",
  "position" : 95,
  "clustering" : [ "item101", 1.5 ],
  "liveness_info" : { "tstamp" :"2020-07-03T23:10:40.326673Z" },
  "cells" : [
{ "name" : "product_code", "value" : "p101" },
{ "name" : "replacements", "value" : ["item101-r", "item101-r2"]
}]}]},

 { "partition" : {
   "key" : [ "CA103" ],
   "position" : 0  },
    "rows" : [ {
    "type" : "row",
    "position" : 74,
    "clustering" : [ "item103", 3.0 ],
    "liveness_info" : {
     "tstamp" : "2020-07-03T23:23:39.440426Z", "ttl" : 30,
      "expires_at" : "2020-07-03T23:24:09Z", "expired" : true
     },
  "cells" : [
{ "name" : "product_code", "value" : "p103" },
{ "name" : "replacements", "value" : ["item101", "item101-r"] }
 ] } ]} ]
```

🟦 What

- Written markers for deleted data in SSTables
- Data with TTL
- UPDATEs and INSERTs with NULL values
- “Overwritten” collections
- Multiple Types
  - Cell, Row, Range, Partition
  - TTL Tombstones (cells and rows)
- Clean during “compaction”

❌ Don’t

- INSERT NULL when you can

✅ Dos

- Always delete as much data as possible in one mutation

Tombstones - Issue #1 = Large Partitions

📘 Issue Definition

- 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

❌ Don't

- "Queueing Pattern" = keep deleting messages in Cassandra

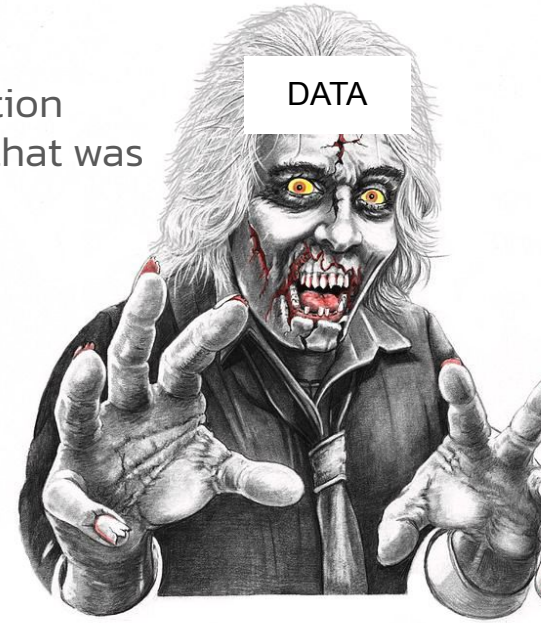
✅ Dos

- Decrease the value of `gc_grace_seconds` (default is 864000 or 10 days)
  - Deleted data and tombstones can be purged during compaction after `gc_grace_seconds`
- Run compaction more frequently
  - `nodetool compact keyspace tablename`

Tombstones Issue #2, Zombie Data

📘 Issue Definition

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



✅ Dos

- Run repairs within the `gc_grace_seconds` and on a regular basis
- `nodetool repair`
- Do not let a node rejoin the cluster after the `gc_grace_seconds`

Agenda

01

Data Modeling
The good, the bad, the ugly

02

Shape your requests up !
It is not “just CQL”

03

Cassandra Graveyard
Tombstones and Zombies

04

Developers Horror Museum
Session, Object Mapping, Frameworks

05

Administration and Operation
Scale like a boss

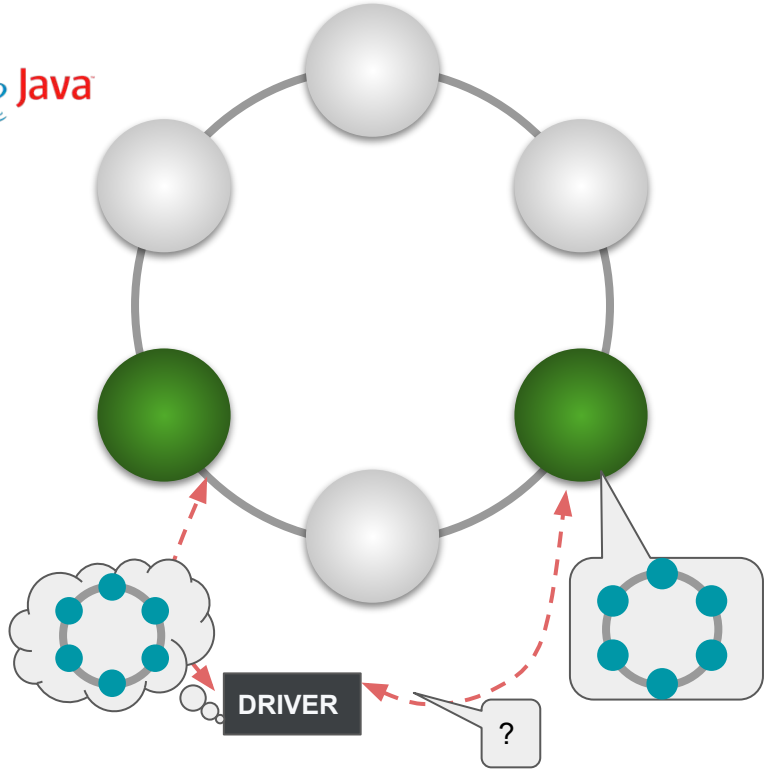
Setup connection with Session (Cluster)

✅ Dos

- Contact points
  - One is enough, consider multiple for first call
  - static-ip or fixed names
  - prioritize seed nodes
- Setup local data center
- Create CqlSession explicitly

❌ Don't

- Multiple keyspaces is a code smell and a can of worms
- Do not stick to default available properties in frameworks



Proper usage of the session

📘 About Session

- Stateful object handling communications with each node
  - Pooling, retries, reconnect, health checks

✅ Dos

- Should be unique in the Application (*Singleton*)
- Should be closed at application shutdown (*shutdown hook*)
- Should be used for fine grained queries (execute)

```
Java:      cqlSession.close();
Python:    session.shutdown();
Node:      client.shutdown();
CSharp:    IDisposable
```

Object Mapping is your closest enemy

❌ DO NOT....

- DO NOT let it create your session (maximum flexibility)
- DO NOT let it generate your schema (metadata are important !)
- DO NOT use OSIV nor Active record but rather Repository
- DO NOT use findAll() = Table Full scan
- DO NOT create large IN queries.
- DO NOT reuse objects from multiple where clause
- DO NOT implement N+1 select, create a new table
- DO NOT stick to simplicity of *Repositories*

✅ DOs....

- DO Prepare your statements
- (Spring Data) : use SimpleCassandraRepository
- (Spring Data) : use CassandraOperations (cqlSession)
- Use Batches, LWT and everything shown before (hidden by ORM)

```
@Repository
public interface TodoRepositoryCassandra extends CassandraRepository<TodoEntity, UUID> {
}
```

```
@Table(value = TodoEntity.TABLENAME)
public class TodoEntity {

 public static final String TABLENAME        = "todos";
 public static final String COLUMN_UID       = "uid";
 public static final String COLUMN_TITLE     = "title";
 public static final String COLUMN_COMPLETED = "completed";
 public static final String COLUMN_ORDER     = "offset";

 @PrimaryKey
 @Column(COLUMN_UID)
 @CassandraType(type = Name.UUID)
 private UUID uid;

 @Column(COLUMN_TITLE)
 @CassandraType(type = Name.TEXT)
 private String title;

 @Column(COLUMN_COMPLETED)
 @CassandraType(type = Name.BOOLEAN)
 private boolean completed = false;

 @Column(COLUMN_ORDER)
 @CassandraType(type = Name.INT)
 private int order = 0;

 public TodoEntity(String title, int offset) {
   this(UUID.randomUUID(), title, false, offset);
 }
}
```

Agenda

01

Data Modeling
The good, the bad, the ugly

02

Shape your requests up !
It is not “just CQL”

03

Cassandra Graveyard
Tombstones and Zombies

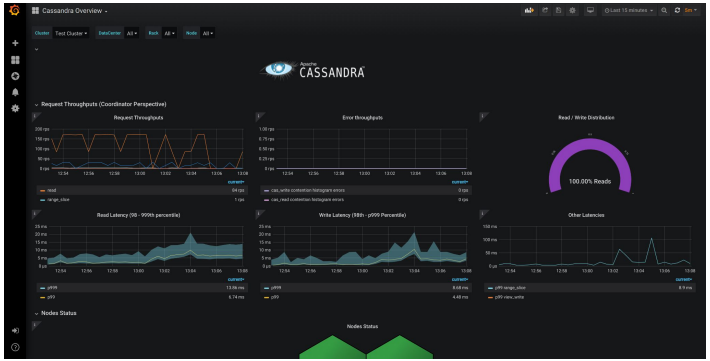
04

Developers Horror Museum
Session, Object Mapping, Frameworks

05

Administration and Operation
Scale like a boss

Administration and Operations



✅ DOs....

- Measure Everything (Disk, CPU, RAM): MCAC, JMX, Virtual tables
- GC pause is the tip of the iceberg, dig into infrastructure
- Run Repairs frequently (incremental) using Reaper
- SNAPSHOTS, Backup and restore
- Do not underestimate security
  - RBAC
  - Encryption at rest
  - Internode communication

❌ DO NOT....

- TUNE BEFORE UNDERSTAND

Thank you!

Sponsored by DataStax