



MASSIV SKALIERBARE UND HOCHVERFÜGBARE DATENHALTUNG MIT APACHE CASSANDRA

Dr. Philipp Walter | Java User Group Saar | 24.01.2023

Einführung

Grundlagen

CQL

Entwicklung

Operations

**Warum und wozu
Cassandra?**

Ziele, Referenzen,
Geschichte

Wozu ist Cassandra
geeignet und wozu
nicht?

**Aufbau von
Cassandra**

Speichermodell und
Architektur für die
geografisch verteilte
Datenhaltung

Replikation und
Konsistenz

Innenleben eines
Knotens beim
Schreiben und Lesen

**Cassandra Query
Language**

Datentypen und
Tabellen erstellen

Daten schreiben, lesen,
löschen

Indizes und
Materialized Views

Transaktionen und
Batches

**Datenmodellierung
und Code**

Migration von Spring
Petclinic auf
Cassandra

Limits, die bei der
Datenmodellierung
berücksichtigt werden
sollten

Cassandra aus Java
ansprechen

**Cassandra produktiv
betreiben**

Installation und
Konfiguration

Administration mit
nodetool

Erste Schritte mit CCM

Weiterführende
Literatur und andere
Quellen

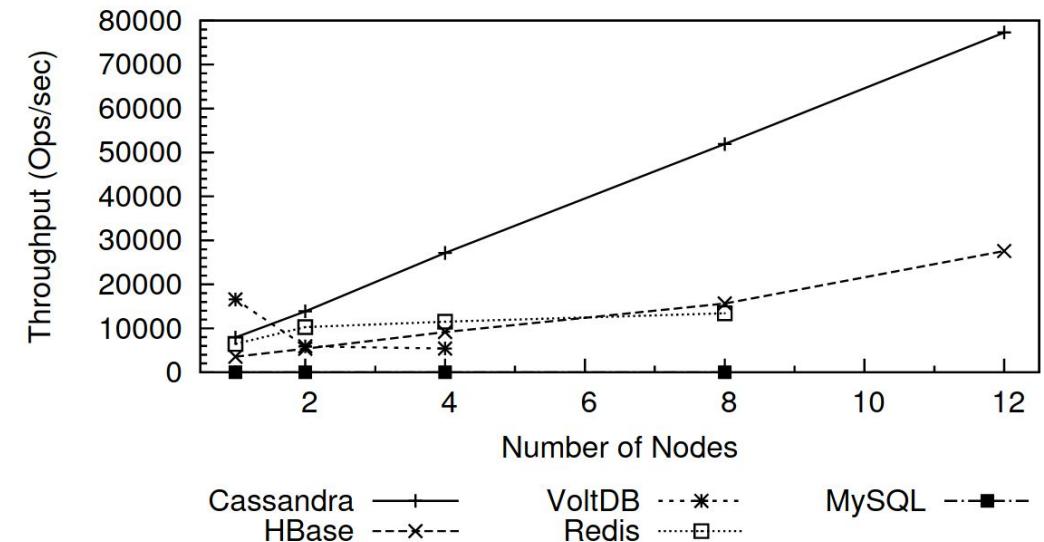
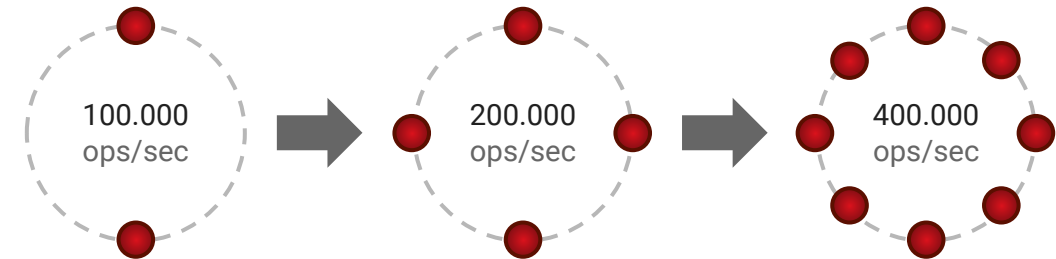


I Einführung

Warum und wozu
Cassandra?

- Welches Problem löst Cassandra?
 - **Petabytes** an **veränderlichen** Daten
 - von **Millionen** simultaner Benutzern auf der Welt
 - verteilt über **Tausende Commodity-Server**
 - an **Hunderten Standorte** auf der Welt

- Was bedeutet das in der Praxis?
 - **Horizontale Skalierbarkeit:** 100% lineare Skalierbarkeit sind ein Muss, sonst ist irgendwann Schluss
 - **Hohe Resilienz:** bei so vielen Maschinen sind tägliche Ausfälle statistisch "normal" und müssen als regulärer Use Case eingeplant werden
 - **Geringer Wartungsaufwand:** auch eine riesige Installation muss von wenigen Admins entspannt verwaltet werden können



- ✓ Strukturierte Daten, d.h. Tabellen
- ✓ Riesige Tabellen mit vielen Zeilen
- ✓ Daten, die mit Point Queries auf Primärschlüsseln abgefragt werden können, z.B.
`SELECT * FROM TABLE WHERE id IN (123, 456)`
- ✓ Gut partitionierbare Daten, z.B. Zeitreihen

Beispiele

- ✓ Kontinuierlich entstehende Daten, z.B. Sensordaten, Logdaten, Sessiondaten, ...
- ✓ Transaktionale Workloads mit kleinen Batches, z.B. Reservierungen von Sitzplätzen – geht mit “lightweight transactions” (compare & swap)

- ✗ Unstrukturierte Daten, z.B. große Blobs, JSON, ...
- ✗ Viele kleine Tabellen mit wenigen Zeilen
- ✗ Daten, die Range Queries über Primärschlüssel erfordern, z.B.
`SELECT * FROM table WHERE id>4711`
- ✗ Schlecht partitionierbare Daten, z.B. Bäume

Beispiele



















- ✗ Komplexe normalisierte Datenbankschemata mit JOINS über viele Tabellen
- ✗ Transaktionale Workloads mit großen Batches, z.B. bei ERP-Software – besser mit MVCC wie bei PostgreSQL abbildbar

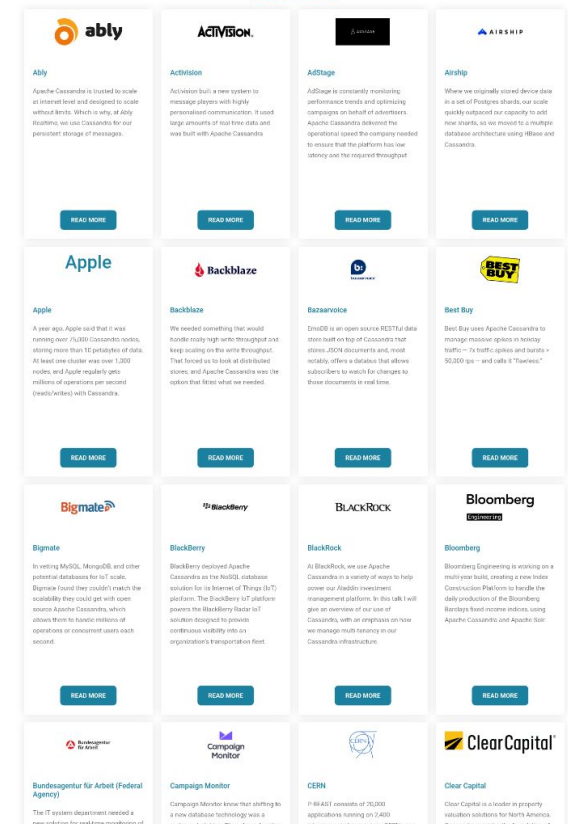
	PostgreSQL	CitusDB	Oracle Exadata	Apache Cassandra	CockroachDB	Google Cloud Spanner
Open Source	✓	✓	✗	✓	✓	✗
Läuft on-premise auf Commodity Hardware	✓	✓	✗	✓	✓	✗
Linear skalierbar	✗	✓	✓	✓	?	✓
Einfach zu betreiben	✓	✗	✗	✓	?	-
ACID-Transaktionen	✓	✓	✓	✗	✓	✓

- Apple: 75.000 Cassandra-Knoten (2015)
- Netflix: 10.000+ Cassandra-Knoten (2018)
- Spotify: 100+ Cluster (2015)



From startups to the largest enterprises, the world runs on Cassandra.

Rang			DBMS	Datenbankmodell	Punkte		
Jan 2023	Dez 2022	Jan 2022			Jan 2023	Dez 2022	Jan 2022
1.	1.	1.	Cassandra 	Wide column	116,31	+1,66	-7,24
2.	2.	2.	HBase	Wide column	39,34	-0,70	-4,65
3.	3.	3.	Microsoft Azure Cosmos DB 	Multi-Model 	37,96	+0,01	-2,08
4.	4.	4.	Datastax Enterprise 	Wide column, Multi-Model 	7,70	-0,93	-2,19
5.	5.	5.	Microsoft Azure Table Storage	Wide column	5,82	+0,20	+0,40
6.	6.	 7.	Accumulo	Wide column	5,70	+0,29	+1,81
7.	 8.	 8.	Google Cloud Bigtable	Multi-Model 	5,47	+0,46	+1,85
8.	 7.	 6.	ScyllaDB 	Wide column, Multi-Model 	5,33	+0,11	+1,42
9.	9.	9.	HPE Ezmeral Data Fabric	Multi-Model 	1,25	-0,15	+0,42
10.	10.	 11.	Amazon Keyspaces	Wide column	0,85	+0,07	+0,28
11.	11.	 10.	Elassandra	Wide column, Multi-Model 	0,52	+0,01	-0,07
12.	12.	12.	Alibaba Cloud Table Store	Wide column	0,27	-0,03	-0,19
13.	13.	13.	SWC-DB	Wide column, Multi-Model 	0,04	-0,01	-0,06



- 2004 Google veröffentlicht das Paper zu **Bigtable**, einem “wide-column store”, der für den Google-Suchindex und andere interne Anwendungen entwickelt wurde
- 2007 Amazon veröffentlicht das Paper zu **Dynamo**, einem verteilten “key-value store”, der für zuverlässige Speicherung bei hoher Schreiblast für interne Anwendungen entwickelt wurde
- 2008 Facebook veröffentlicht die erste Version von **Cassandra**, die intern für die Inbox-Suche auf Facebook entwickelt wurde, als Open Source
- 2009 Cassandra wird **Apache Cassandra**
- 2010 Apache Cassandra wird zum **Top-Level-Projekt** erhoben
DataStax wird gegründet und veröffentlicht mit **DataStax Enterprise (DSE)** ein “Enterprise-Cassandra”
- 2011 Apache Cassandra **1.0**
- 2013 Apache Cassandra **2.0**
- 2015 Apache Cassandra **3.0**
- 2021 Apache Cassandra **4.0**

- Cassandra ist auch nützlich, wenn man kleinere Datenmengen hat
 - Open Source
 - Einfach zu installieren und praktisch wartungsfrei (1 Java-Prozess je Knoten)
 - Einfaches Datenmodell (wide-column store)
 - Exzellent ins Ökosystem integriert (Treiber, Spring, Quarkus, IntelliJ, ...)
- Cassandra kann einfach lokal betrieben werden
 - auch als "echter" Cluster mit mehreren Servern



II Grundlagen

Aufbau von Cassandra

Jahr	Monat	Anrufer	Zeit	Nummer	Dauer	} Datensatz (Row)
2022	11	+49171937468	15.11.2022 15:36:49	+4915184727	748s	
2022	11	+49176648300	20.11.2022 19:42:18	+4916037478	44s	
2022	12	+49160123987	09.12.2022 11:00:00	+49172456987	634s	
2022	12	+49151746328	24.12.2022 19:10:00	+49172836492	112s	
2022	12	+49160123987	30.12.2022 09:42:00	+49154828374	89s	
2023	01	+49152947490	01.01.2023 08:12:00	+49170847236	234s	
2023	01	+49171948368	02.01.2023 14:57:00	+49171938475	469s	
					} Wert (Key-Value Pair, Column)	

Jahr	Monat	Anrufer	Zeit	Nummer	Dauer	
2022	11	+49171937468	15.11.2022 15:36:49	+4915184727	748s	} Datensatz (Row)
2022	11	+49176648300	20.11.2022 19:42:18	+4916037478	44s	
2022	12	+49160123987	29.12.2022 11:00:00	+49172456987	634s	} Partition
2022	12	+49151746328	30.12.2022 19:10:00	+49172836492	112s	
2022	12	+49160123987	31.12.2022 09:42:00	+49154828374	89s	
2023	01	+49152947490	01.01.2023 08:12:00	+49170847236	234s	
2023	01	+49171948368	02.01.2023 14:57:00	+49171938475	469s	

} **Partition Key**
} **Wert**
 (Key-Value Pair, Column)

Jahr	Monat	Anrufer	Zeit	Nummer	Dauer	
2022	11	+49171937468	15.11.2022 15:36:49	+4915184727	748s	} Datensatz (Row)
2022	11	+49176648300	20.11.2022 19:42:18	+4916037478	44s	
2022	12	+49151746328	30.12.2022 19:10:00	+49172836492	112s	} Partition
2022	12	+49160123987	29.12.2022 11:00:00	+49172456987	634s	
2022	12	+49160123987	31.12.2022 09:42:00	+49154828374	89s	
2023	01	+49152947490	01.01.2023 08:12:00	+49170847236	234s	
2023	01	+49171948368	02.01.2023 14:57:00	+49171938475	469s	

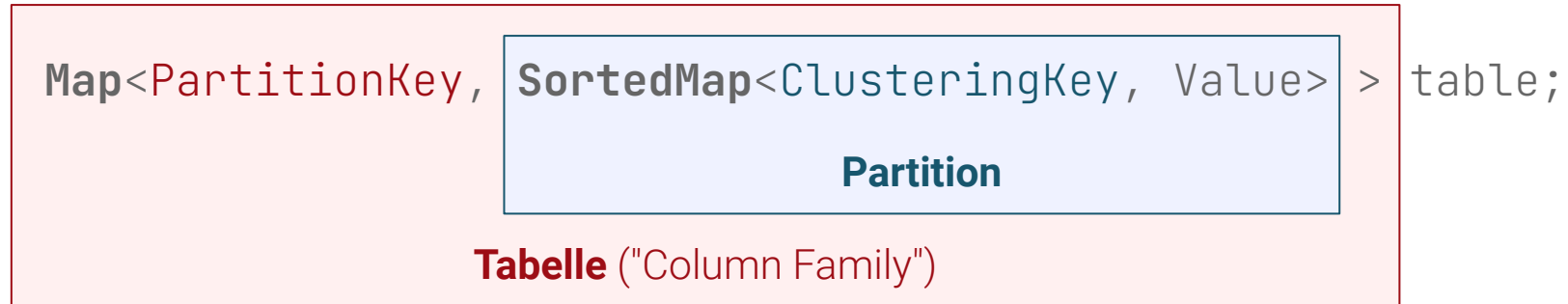
Partition Key

Clustering Key
 (Clustering Columns)

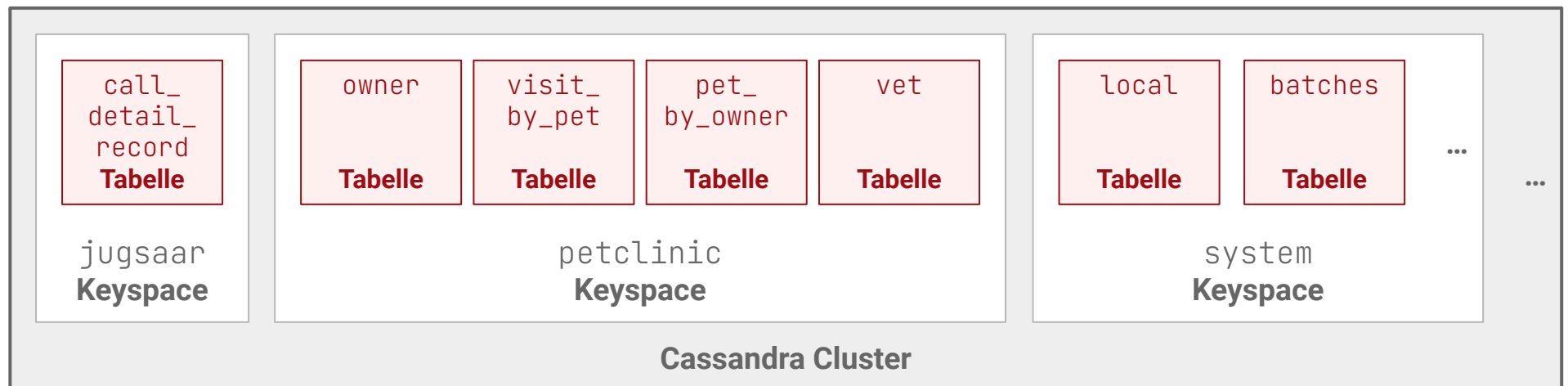
Wert
 (Key-Value Pair, Column)

Primary Key

- **Tabellen** werden verteilt als **Distributed Hash Tables** (DHT) gespeichert



- Tabellen sind immer Teil von **Keyspaces**



1. Keyspace anlegen

```
cqlsh> CREATE KEYSPACE jugsaar WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor': 1};
```

2. Tabelle anlegen

```
cqlsh> CREATE TABLE jugsaar.call_detail_record (  
...   year INT, month INT, caller TEXT, time TIMESTAMP, called TEXT, duration_seconds INT,  
...   PRIMARY KEY ((year, month), caller, time)  
... ) WITH CLUSTERING ORDER BY (caller ASC, time ASC);
```

3. Schreiben und Lesen

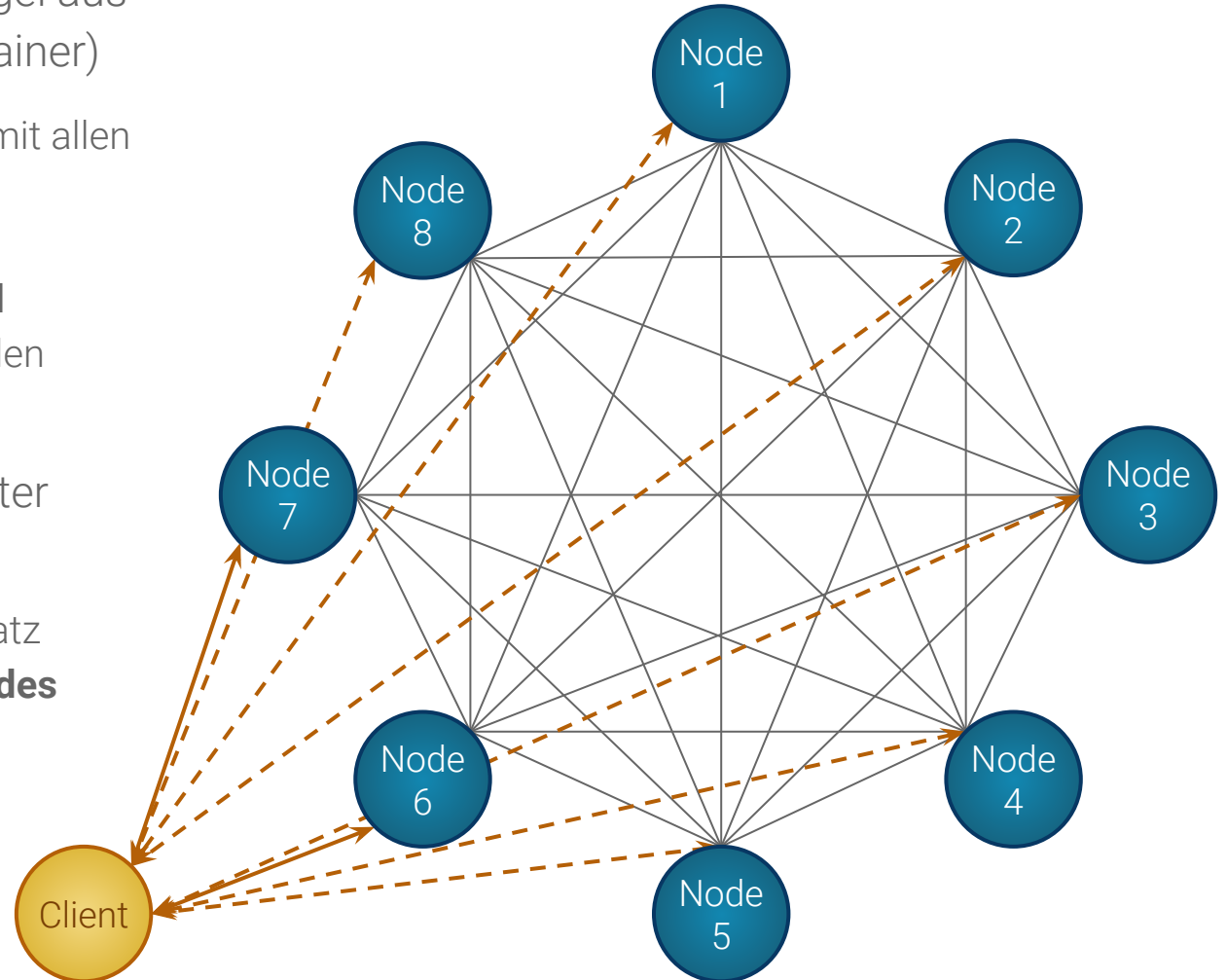
```
cqlsh> INSERT INTO jugsaar.call_detail_record (year, month, caller, time, called, duration_seconds) VALUES ...
```

```
cqlsh> SELECT * FROM jugsaar.call_detail_record WHERE year=2022 AND month=12;
```

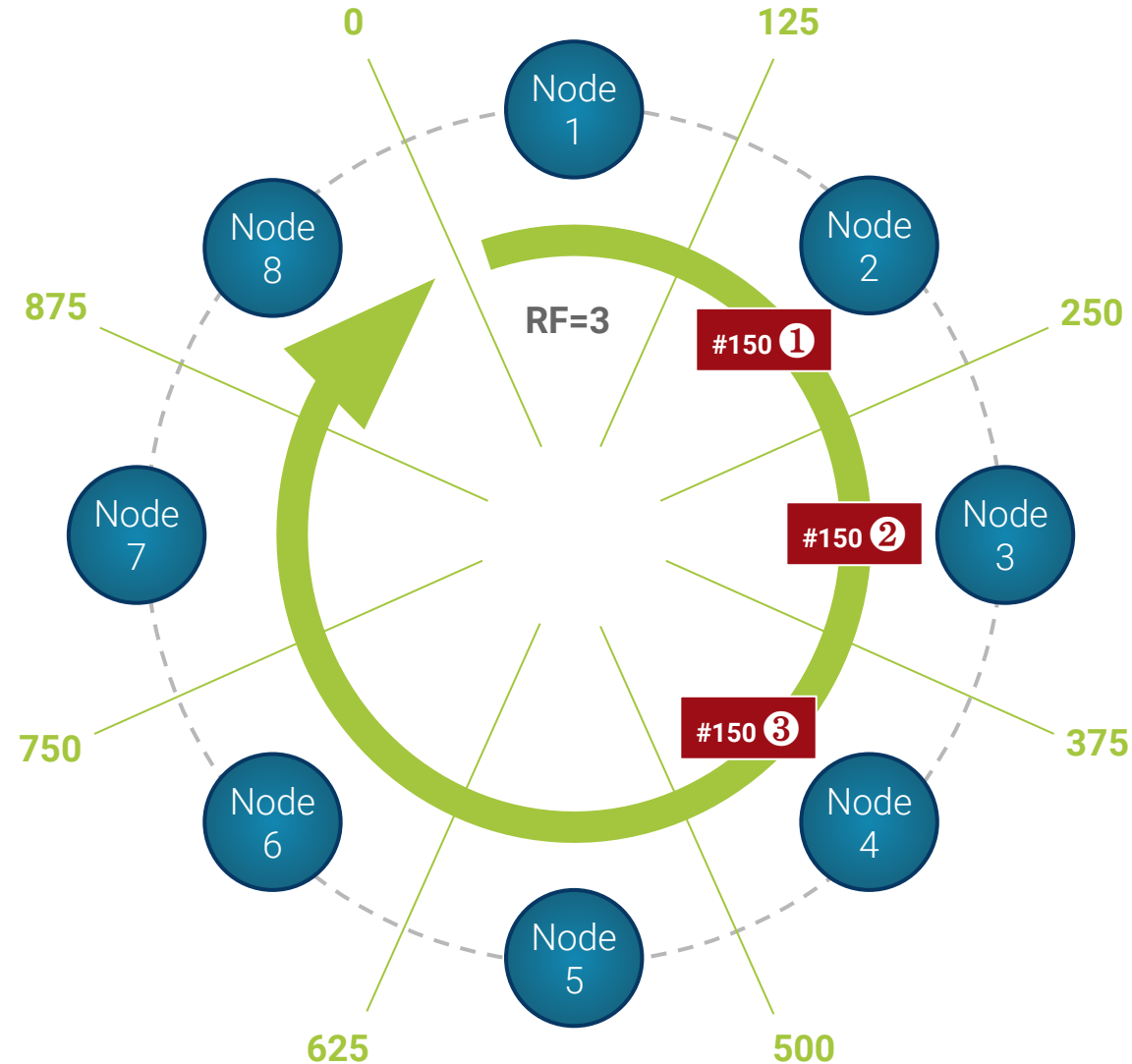
year	month	caller	time	called	duration_seconds
2022	12	+49151746328	2022-12-24 18:10:00.000000+0000	+49172836492	112
2022	12	+49160123987	2022-12-09 10:00:00.000000+0000	+49172456987	634
2022	12	+49160123987	2022-12-30 08:42:00.000000+0000	+49154828374	89

(3 rows)

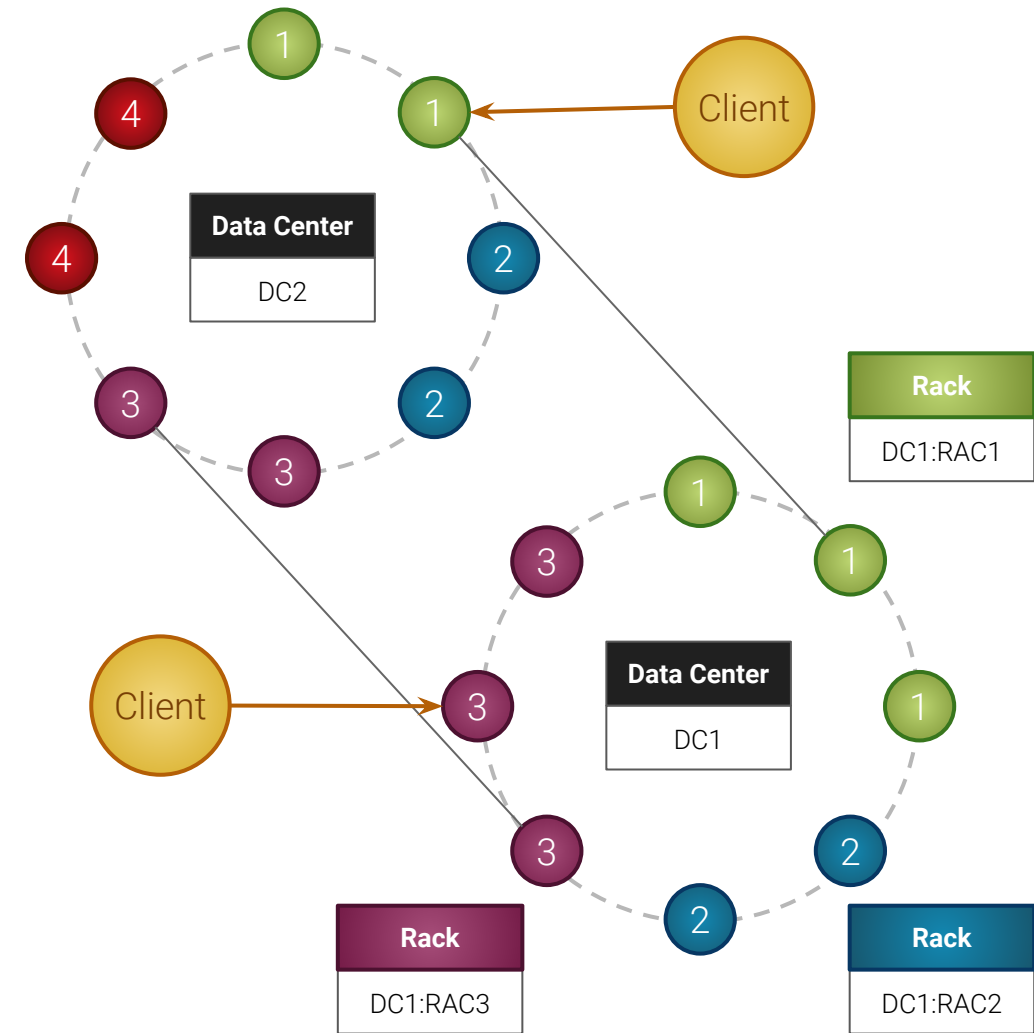
- Ein **Cassandra-Cluster** besteht in der Regel aus mindestens 3 **Knoten** (Server, VMs, Container)
 - Jeder Knoten ist immer über TCP direkt mit allen anderen Knoten verbunden
 - Alle Knoten informieren sich in Echtzeit untereinander über das **Gossip-Protokoll**
 - Jeder Knoten kennt so immer den aktuellen Zustand des kompletten Clusters
- **Clients** sprechen über TCP mit dem Cluster
 - Als Treiber kommen i.d.R. die DataStax-Open-Source-Treiber zum Einsatz
 - Jeder Client kontaktiert erst die **Seed Nodes** und besorgt sich die Cluster-Topologie
 - Dann wird eine **Session** mit weiteren Verbindungen zu allen Knoten erstellt
 - Abfragen laufen anschließend über **CQL** direkt an die Knoten



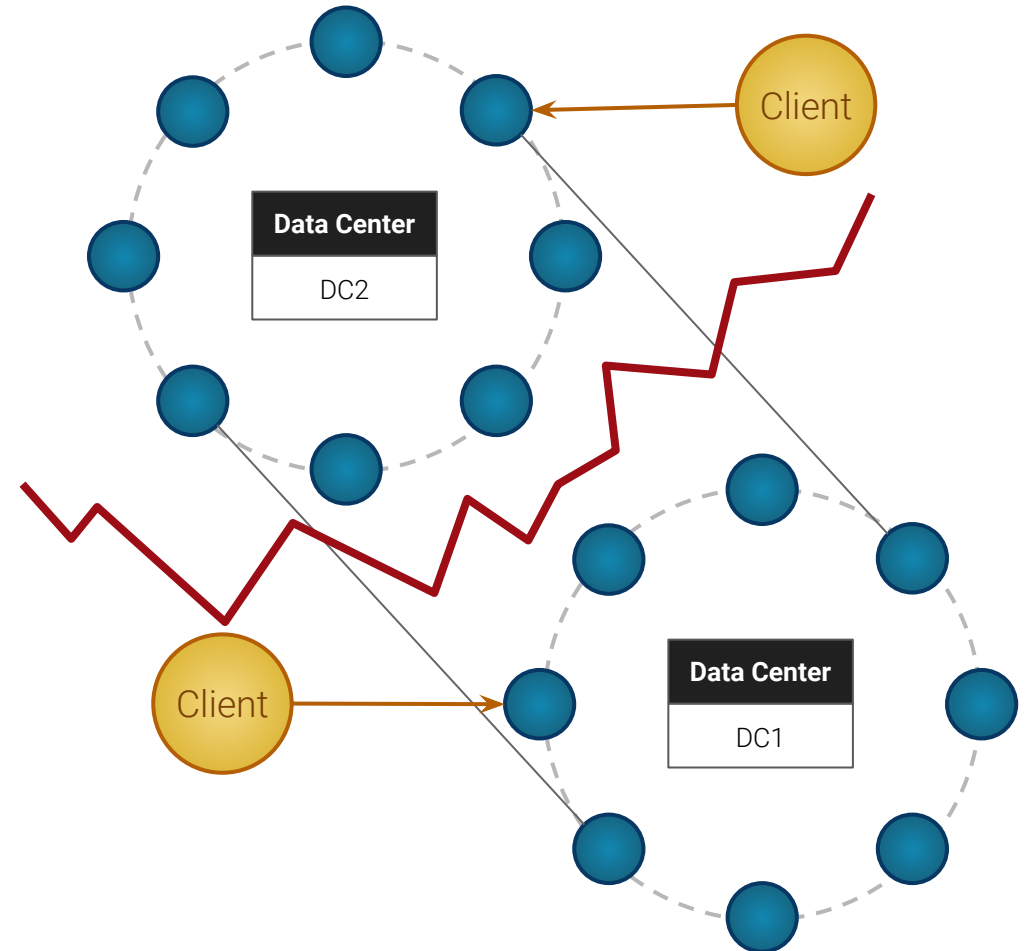
- Datensätze eines Keyspace werden in Cassandra über den gesamten Cluster verteilt
 - Für jeden **Partition Key** wird ein **Token** berechnet
 - Die Daten werden nach ihren Tokens gleichmäßig auf alle Knoten verteilt
 - **Consistent Hashing:** durch neue oder wegfallende Knoten ändern sich nicht alle Zuordnungen
- Von jedem Datensatz werden mehrere **Replicas** gespeichert
 - Anzahl aller Exemplare = **Replication Factor** (RF) des Keyspace
 - Replicas werden standardmäßig auf die nachfolgenden Knoten im Ring platziert
 - Bei Ausfall eines Knotens bedienen ein oder mehrere Nachfolger die Abfrage



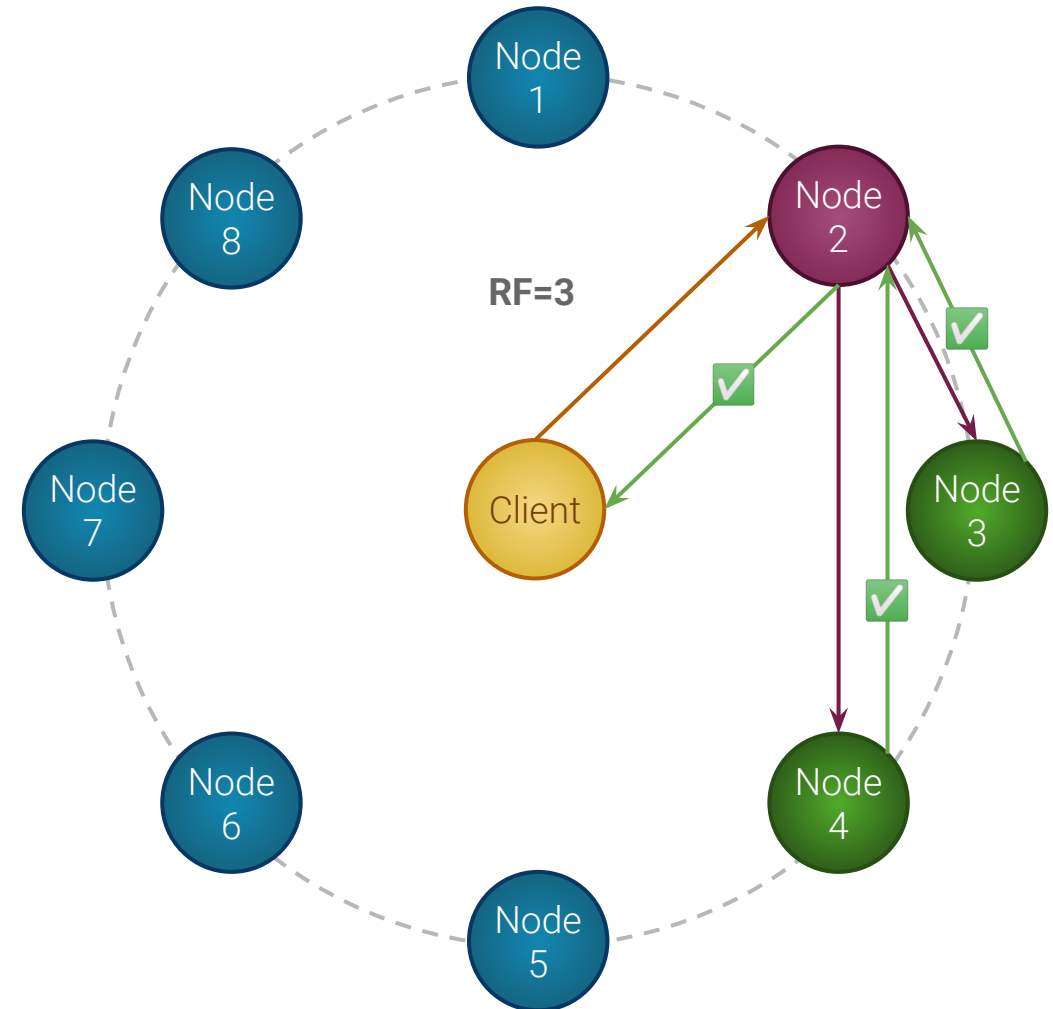
- Replikation über verschiedene Standorte
 - Jeder Standort ist ein **Data Center**, in dem bei Bedarf alle Daten redundant und aktuell sind
 - In jedem Data Center gibt **Racks**, in denen die Knoten installiert sind
 - Clients können bevorzugt ihr **lokales** Data Center kontaktieren, das am schnellsten erreichbar ist
 - In jedem Data Center ist **Lesen und Schreiben** aller Datensätze möglich
- Die Replikation erledigt Cassandra automatisch
 - Jeder Keyspace kann in jedem Data Center einen anderen Replication Factor haben (auch 0)
 - Die Replicas werden in jedem Data Center nach Möglichkeit auf verschiedene Racks verteilt



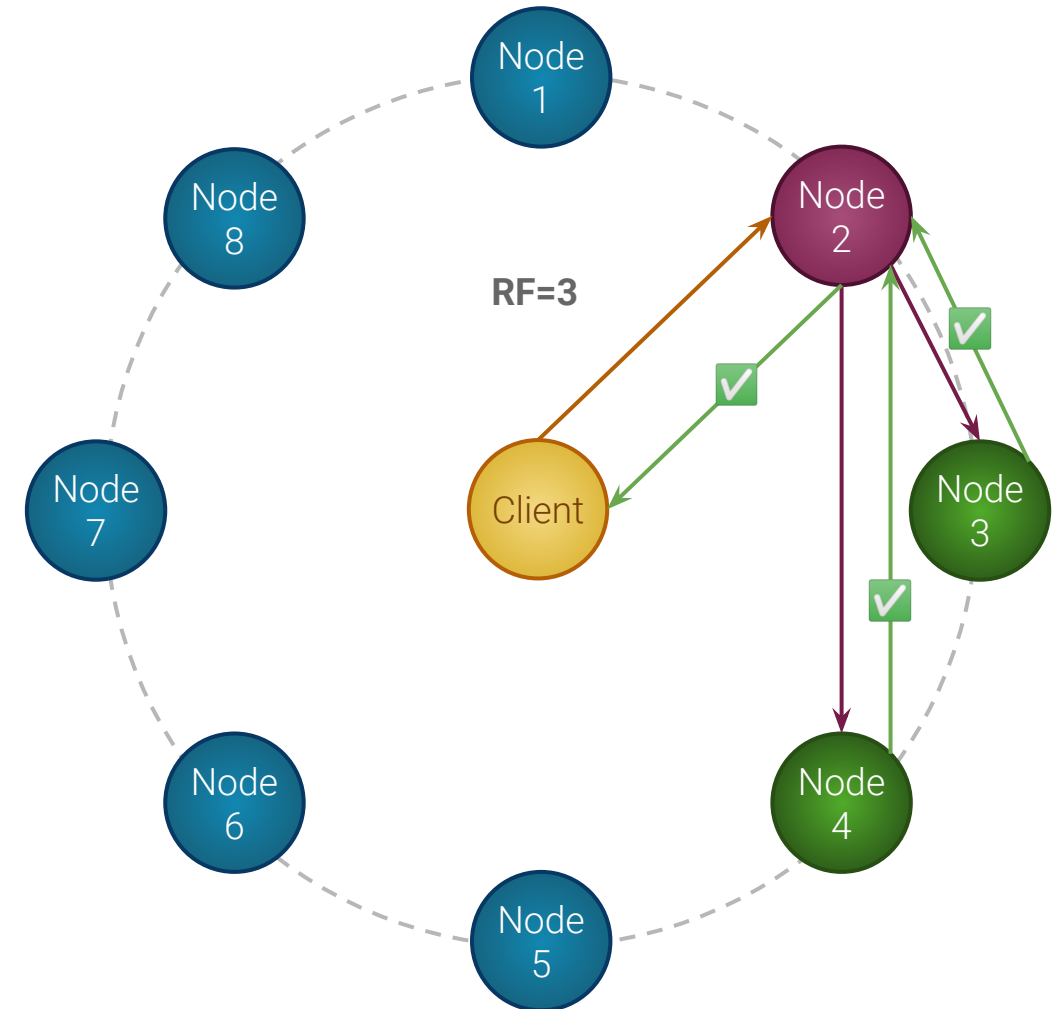
- Das CAP-Theorem besagt, dass eine Datenbank nur zwei der drei Eigenschaften erfüllen kann:
 - **Consistency** = alle Replicas sind immer auf einem aktuellen konsistenten Stand
 - **Availability** = es können immer Abfragen beantwortet werden, es gibt keine Downtimes
 - **Partition Tolerance** = auch wenn die Verbindung zwischen einem Teil der Knoten verloren geht, kann das System weiterarbeiten
- Durch seine Replicas kann Cassandra "AP"
- Wenn ein Knoten, ein Rack oder ein Data Center ausfällt, können die übrigen Knoten noch Abfragen beantworten



- Ein Client wählt einen Knoten als **Coordinator** und schickt ihm sein UPDATE
 - Der Coordinator kontaktiert alle Knoten mit den notwendigen Replicas
 - Die Knoten schreiben die Replicas und melden OK zurück
- Das **Consistency Level** bestimmt, wann der Coordinator selbst ein OK zurück gibt:
 - ONE, TWO, THREE = nach 1/2/3 Replicas
 - QUORUM = nach mehr als die Hälfte der Replicas
 - ALL = erst nach allen Replicas
 - LOCAL_ONE, LOCAL_QUORUM = wie ONE und QUORUM, nur beschränkt auf das aktuelle Data Center
 - EACH_QUORUM = QUORUM in jedem Data Center
 - ANY = nach 1 Replica oder wenn ein Hint gespeichert wurde (kommt noch)



- Ein Client wählt einen Knoten als **Coordinator** und schickt ihm sein SELECT
- Das **Consistency Level** bestimmt, wie viele Replicas der Coordinator abfragt:
 - ONE, TWO, THREE = die schnellsten 1/2/3 Knoten
 - QUORUM = die schnellsten $RF/2+1$ Knoten
 - ALL = alle Knoten
- Tunable Consistency
 - Nummer sicher:
 $\text{Schreib-Level} + \text{Lese-Level} > RF$
 - Schnell lesen:
 $ALL(3) + ONE(1) = 4 > 3$ ✓
 - Schnell schreiben:
 $ONE(1) + ALL(3) = 4 > 3$ ✓
 - Ausgewogen:
 $QUORUM(2) + QUORUM(2) = 4 > 3$ ✓



- <https://www.ecyrd.com/cassandrascalculator/>

Cassandra Parameters for Dummies

This simple form allows you to try out different values for your [Apache Cassandra](#) cluster and see what the impact is for your application.

Cluster size Replication Factor Write Level Read Level

Your reads are **consistent**

"Consistent" means that for this particular Read/Write level combo, all nodes will "see" the same data. "Eventually consistent" means that you might get old data from some nodes and new data for others until the data has been replicated across all devices. The idea is that this way you can increase read/write speeds and improve tolerance against dead nodes.

You can survive the loss of **1 node** without impacting the application.

How many nodes can go down without application noticing? This is a lower bound - in large clusters, you could lose more nodes and if they happen to be handling different parts of the keyspace, then you wouldn't notice either.

You can survive the loss of **1 node** without data loss.

How many nodes can go down without physically losing data? This is a lower bound - in large clusters, you could lose more nodes and if they happen to be handling different parts of the keyspace, then you wouldn't notice either.

You are really reading from **2 nodes** every time.

The more nodes you read from, more network traffic ensues, and the bigger the latencies involved. Cassandra read operation won't return until at least this many nodes have responded with some data value.

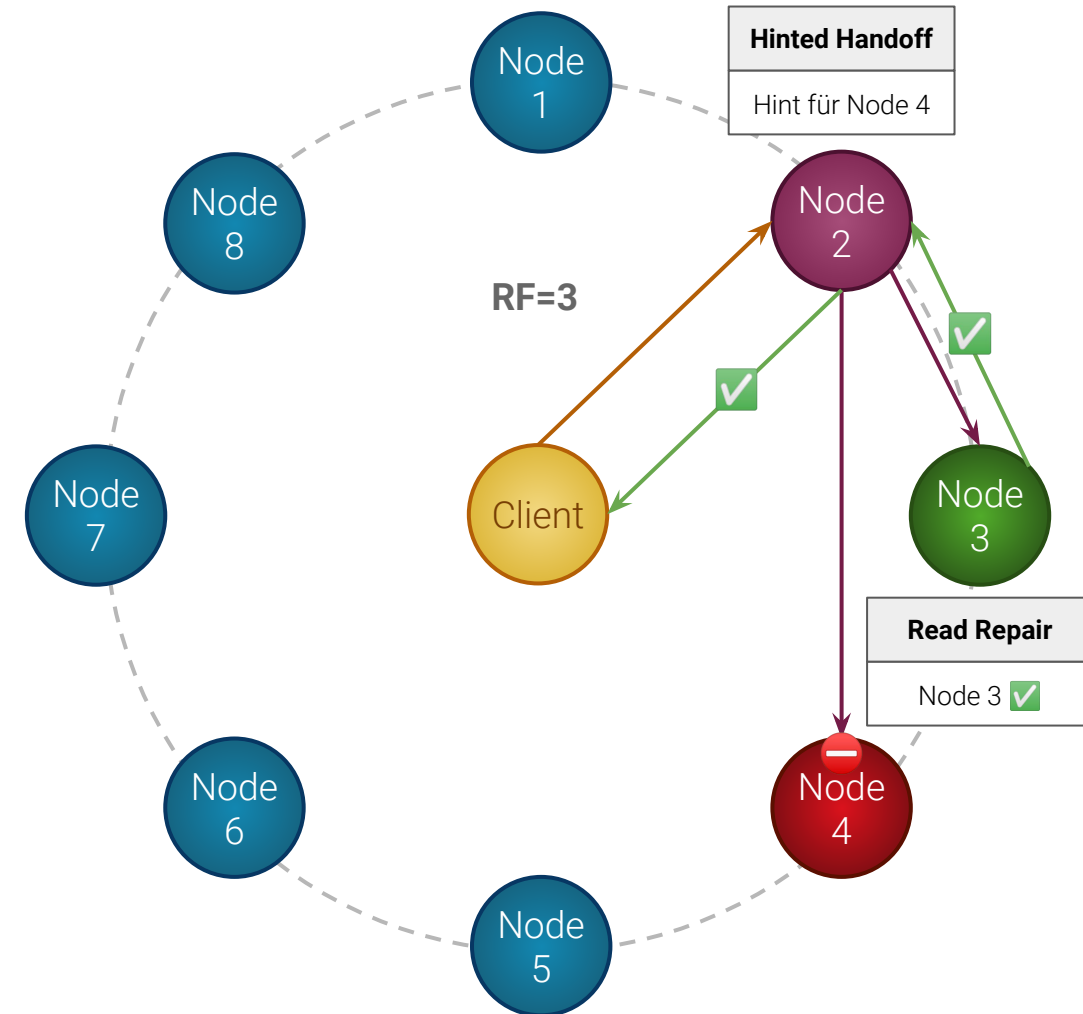
You are really writing to **2 nodes** every time.

The more nodes you write to, more network traffic ensues, and the bigger the latencies involved. Cassandra write operation won't return until at least this many nodes have acknowledged receiving the data.

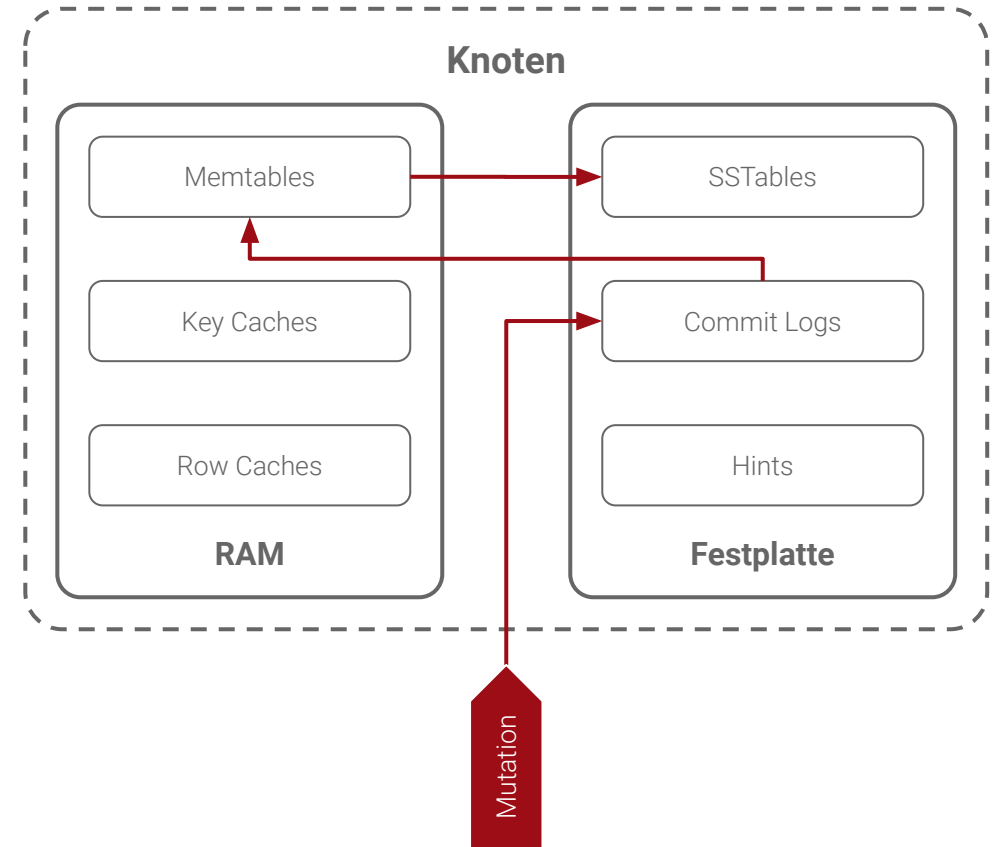
Each node holds **38%** of your data.

The bigger your cluster is, the more the data gets distributed across your nodes. If you are using the RandomPartitioner, or are very good at distributing your keys when you use OrderedPartitioner, this is how much data each of your nodes has to handle. This is also how much of your keyspace becomes inaccessible for each node that you lose beyond the safe limit, above.

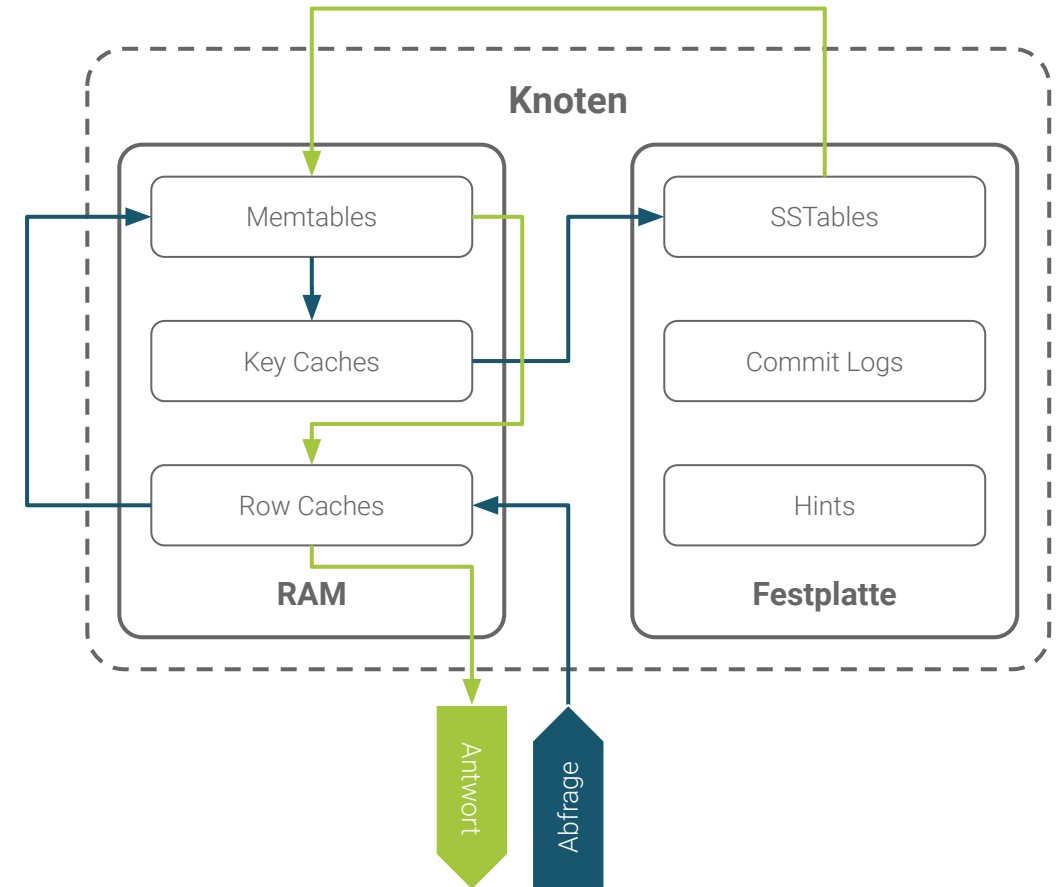
- **Entropie:** nicht alle Replicas sind immer konsistent auf dem letzten Stand
 - **Last Write Wins:** jede Änderung hat einen **Timestamp**, bei mehreren Änderungen wird die neueste Version genommen
 - **Read Repair:** wenn der Coordinator beim Lesen merkt, dass ein Knoten alte Daten hat, korrigiert er die auf den neuesten Stand
 - **Hinted Handoff:** Coordinator sammelt Änderungen für einen Offline-Knoten und überträgt sie, wenn der Knoten wieder da ist
 - **Anti-Entropy Repair:** regelmäßig alle X Tage sollten die kompletten Daten im Cluster einmal aktualisiert werden
- **Eventual Consistency:** zum Schluss sind alle Replicas konsistent



- Schreibzugriffe (**Mutations**) speichert die **Storage Engine** des Cassandra-Service
 1. im **Commit Log** auf Platte, damit sie bei einem Ausfall nicht verloren gehen,
 2. dann in **Memtables** im RAM, bis die oder das Commit Log zu groß werden, und
 3. schließlich als **SSTables** auf Festplatte.
- Eine SSTable ist ein Verzeichnis im Dateisystem mit den Daten einer einzelnen Tabelle und mehreren Dateien:
 1. komprimierte, nach Partition und Clustering Keys sortierte **Rohdaten**
 2. **Index**, an welchen Positionen in der Datei Partitionen beginnen
 3. **Bloom-Filter** über die Partition Keys
 4. Daten-Samples, Statistiken, Checksummen



- Lesezugriffe werden nacheinander zu beantworten versucht mit
 1. einem Eintrag in den **Row Caches**,
 2. einem Eintrag in einer **Memtable**,
 3. und schließlich aus den **SSTables**.
- Der Zugriff auf die SSTables wird beschleunigt
 1. durch Bloom-Filter, die **fast negative lookups** ermöglichen,
 2. durch **Key Caches**, die die Position eines Partition Keys in einer SSTable speichern, sowie
 3. durch den **Index** jeder SSTable.
- Die gefundenen Datensätze werden anschließend
 1. in den Memtables gespeichert,
 2. im Row Cache eingetragen, und
 3. an den Client oder Coordinator geliefert.



- Mit jeder Memtable, die auf Platte geschrieben wird, kommt mindestens eine weitere SSTable dazu
 - Die enthält nur die zuletzt gelesenen oder geänderten Daten einer Tabelle
 - Alte Datensätze in alten SSTables können noch aktuell sein
- SSTables müssen daher regelmäßig konsolidiert werden
 - Bei **Compaction** einer Tabelle werden ihre aktuellen Daten in neuen SSTables gesammelt
 - Ein Sonderfall aktueller Werte sind **Tombstones**, mit denen Werte zum Löschen überschrieben werden, und die nach 10 Tagen gelöscht werden
 - Cassandra führt regelmäßig automatische **Minor Compactions** auf einem Teil der SSTables aus
 - Manuell kann auch eine vollständige **Major Compaction** aller SSTables angestoßen werden
- Compaction-Strategien
 - **Size Tiered Compaction** (Standard): gruppiert SSTables ähnlicher Größe und compacted die Gruppe mit den meisten Lesezugriffen
 - **Leveled Compaction**: hält SSTables überschneidungsfrei, sodass beim Lesen immer nur wenige SSTables angefasst werden müssen
 - **Time Window Compaction**: speziell für unveränderliche Zeitreihendaten mit einem Ablaufdatum, z.B. Logdaten



III CQL

—
Cassandra Query Language



- An SQL angelehnte Abfragesprache
 - benutzt dieselbe Syntax und dieselben Abstraktionen: Tabellen, Datensätze, Spalten
 - in bin/ mitgeliefertes Standard-CLI ist **cqlsh**
 - Vollständige Doku: <https://cassandra.apache.org/doc/latest/cassandra/cql/definitions.html>
- Je Cassandra-Version werden verschiedene Versionen des **Native Protocol** und **CQL** unterstützt:

Cassandra	Native Protocol	CQL Specs
4.x	v5	3.4.x
3.x	v4	3.3.x
2.2.x	v4	3.1.x
2.1.x	v3	3.1.x
2.0.x	v2	3.1.x
1.2.x	v1	3.0.x

```
$ cqlsh localhost 9042
Connected to Test Cluster at 127.0.0.1:9042
[cqlsh 6.1.0 | Cassandra 4.1.0 | CQL spec 3.4.6 | Native protocol v5]
Use HELP for help.
cqlsh>
```


Cassandra-Typ	Java-Typ	CQL-Literale
int	int	1234567890
bigint	long	1234567890123456789
varint	java.math.BigInteger	123456789012345678901234567890
smallint	short	12345
tinyint	byte	123
float	float	1234567890, 1.234567890
double	double	1234567890123456789, 1.234567890123456789
decimal	java.math.BigDecimal	123456789012345678901234567890, 1.23456789012345678901234567890
ascii, varchar, text	java.lang.String	'asdf' (ascii wird als ASCII, varchar und text als UTF-8 gespeichert)
boolean	boolean	True, False, true, false
blob	java.nio.ByteBuffer	0xCAFEBAFE, 0xdeadbeef
uuid, timeuuid	java.util.UUID	2d8c4740-89b4-11ed-9973-97ed4b4fea1f, now() (uuid ist eine beliebige UUID, timeuuid eine v1-UUID für konfliktfreie Primärschlüssel)
counter	long	Nur hoch- und runterzählen erlaubt: UPDATE table SET col=col+4711

Cassandra-Typ	Java-Typ	CQL-Ausdrücke
date	<code>java.time.LocalDate</code>	'2022-12-30' oder Tage seit 1.1.1970
time	<code>java.time.LocalTime</code>	'19:00:00', '19:00:00.123456789' oder Nanosekunden seit 0:00 Uhr
timestamp	<code>java.time.Instant</code>	'2022-12-30 19:00:00', '19:00:00.123456789' oder Millisekunden seit 1.1.1970 0:00 Uhr
duration	<code>CqlDuration</code>	89h4m48s, 3y4mo, 1234ms, ...

- Alle Zeiten werden in Cassandra ohne Zeitzone gespeichert – idealerweise hält man sie in UTC
- `cqlsh` zeigt Zeiten in der lokal auf dem Client konfigurierten Zeitzone an, man kann aber UTC erzwingen mit

```
TZ=UTC cqlsh localhost 9042
```

- Von Instant werden in timestamp nur die Millisekunden gespeichert, also `instant.truncatedTo(ChronoUnit.MILLIS)`

Cassandra-Typ	Java-Typ	CQL-Ausdrücke
map<a,b>	java.util.Map<A,B>	{123: 'asdf'} oder Update einzelner Elemente mit UPDATE table SET m[123]='asdf'
set<a>	java.util.Set<A>	{123,456}
list<a>	java.util.List<A>	[123,456]
tuple<a,b,c>	TupleValue	(123,'asdf',true)

- Zwei Speicherformate
 - map<list<int>> speichert seine Einträge in den SSTables als einzelne Key-Value-Paare, Einträge können daher einzeln geändert werden
 - **frozen**<list<int>> speichert die Einträge serialisiert als Blob, kann nur als Ganzes gelesen und ersetzt werden
- Collections sind für kleine, überschaubare Datenmengen gedacht
 - Alles, was theoretisch unbegrenzt wachsen kann (alle Nachrichten eines Users etc.) sollte in eine eigene Tabelle

- User-Defined Types werden feldweise und pro Keyspace definiert

```
cqlsh> CREATE TYPE jugsaar.version (  
...     major SMALLINT,  
...     minor SMALLINT,  
...     patch SMALLINT,  
...     prerelease TEXT,  
...     build BLOB  
... );
```

- "Frozen" können sie überall wie BLOB eingesetzt werden, z.B. in Collections als Werte

```
cqlsh> CREATE TABLE jugsaar.software_map (  
...     name      TEXT,  
...     releases  MAP<TIMESTAMP, FROZEN<version>>,  
...     PRIMARY KEY (name)  
... );
```

```
cqlsh> UPDATE jugsaar.software_map SET releases['2022-12-13 08:25'] = {major:4, minor:1, patch: 0}  
... WHERE name='Apache Cassandra';
```

- Der Keyspace definiert die Replikation für alle Tabellen darin

```
cqlsh> CREATE KEYSPACE jugsaar WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor': 1};
```

- Tabellen definieren die Struktur von Keys und Werten

```
cqlsh> CREATE TABLE jugsaar.call_detail_record (  
    year            INT,  
    month           INT,  
    caller          TEXT,  
    time            TIMESTAMP,  
    called          TEXT,  
    duration_seconds INT,  
    PRIMARY KEY ((year, month), caller, time)  
) WITH CLUSTERING ORDER BY (caller ASC, time ASC);
```

- Spalten haben nur Namen und Datentypen
- Ihre Rolle als Partition Key, Clustering Key oder Wert wird in PRIMARY KEY festgelegt
- Die CLUSTERING ORDER bestimmt, wie die Daten in den SSTables sortiert werden, damit man idealerweise immer sequenziell lesen kann (ASC/ASC hier eigentlich redundant, weil default)

```
cqlsh:jugsaar> SELECT * FROM call_detail_record;
```

year	month	caller	time	called	duration_seconds
2022	12	+49160123987456	2022-12-30 19:00:00.000000+0000	+49172456987123	634

(1 rows)

- Nur einfaches SELECT ... FROM ... WHERE ...
 - kein JOIN
 - kein Sub-SELECT
- WHERE-Klauseln müssen dem Map-Charakter Rechnung tragen:
 - Partition Keys können nur mit = und IN auf allen Feldern gefiltert werden
 - Clustering Key können mit =, <, > und IN gefiltert werden, ABER ...
 - ... innerhalb einer Partition kann nur eine (1) zusammenhängende Folge von Datensätzen selektiert werden
 - Partition Keys kann man mit SELECT DISTINCT auflisten
 - ALLOW FILTERING erlaubt Ausnahmen, die **langsam und teuer** sind

Statt ALLOW FILTERING sollte man immer das Schema anpassen!

■ INSERT und UPDATE machen dasselbe

```
cqlsh:jugsaar> INSERT INTO call_detail_record (year, month, caller, time, called, duration_seconds)
... VALUES (2022, 12, '+49160123987456', '2022-12-30 19:00:00', '+49172456987123', 634);
```

ist äquivalent zu

```
cqlsh:jugsaar> UPDATE call_detail_record SET called='+49172456987123', duration_seconds=634
... WHERE year=2022 AND month=12 AND caller='+49160123987456' AND time='2022-12-30 19:00:00';
```

- INSERT geht auch, wenn schon ein Wert existiert
- UPDATE geht auch, wenn noch kein Wert existiert

■ Timestamp und Lebensdauer eines Werts können manuell gesetzt werden

```
cqlsh:jugsaar> UPDATE call_detail_record USING TIMESTAMP 1672585200
... SET called='+49172456987123', duration_seconds=634
... WHERE year=2022 AND month=12 AND caller='+49160123987456' AND time='2022-12-30 19:00:00';
```

```
cqlsh:jugsaar> UPDATE call_detail_record USING TTL 1000
... SET called='+49172456987123', duration_seconds=634
... WHERE year=2022 AND month=12 AND caller='+49160123987456' AND time='2022-12-30 19:00:00';
```


- **DELETE** löscht einzelne Werte oder ganze Partitionen

```
cqlsh:jugsaar> DELETE FROM call_detail_record WHERE year=2022 AND month=11;
```

- **TRUNCATE** leert ganze Tabellen

```
cqlsh:jugsaar> TRUNCATE call_detail_record;
```

- **DROP** löscht die komplette Tabelle samt Schema, Indizes etc.

```
cqlsh:jugsaar> DROP call_detail_record;
```

■ Secondary Indexes

```
cqlsh> CREATE INDEX callees ON call_detail_record (called);
```

- erzeugt intern eine "inverse" Tabelle mit Werten als Keys:
- gut auf Spalten mit Werten, die in mehreren, aber nicht einem Großteil der Zeilen vorkommen
- schlecht auf Spalten mit Werten, die sich oft ändern, da sich jeder Schreibzugriff vervielfacht
- alternativ eigene denormalisierte Tabelle erstellen und aktuell halten

■ SSTable Attached Secondary Indexes (SASI)

```
cqlsh:jugsaar> CREATE CUSTOM INDEX callee_contains ON call_detail_record (called)  
... USING 'org.apache.cassandra.index.sasi.SASIIndex'  
... WITH OPTIONS = {'mode': 'CONTAINS'};
```

- Index wird nicht in einer Tabelle, sondern als weitere Datei einer SSTable gespeichert – sehr schnell beim Lesen
- kann mit mode auf die benötigten Queries angepasst werden: CONTAINS erlaubt LIKE '%infix%', PREFIX erlaubt LIKE 'prefix%', SPARSE für Range Queries über Spalten mit vielen unterschiedlichen Werten
- mit analyzer_class kann Text z.B. für Volltextsuche indiziert werden, LIKE 'jugsaar' findet dann z.B. auch alle ähnlichen Schreibweisen
- experimentelles Feature, das (noch) in cassandra.yaml aktiviert werden muss

- **Materialized Views** denormalisieren die Daten bestehender Tabellen automatisch:

```
cqlsh:jugsaar> CREATE MATERIALIZED VIEW callers_by_callee
...           AS SELECT year, month, time, caller, called FROM call_detail_record
...           WHERE year IS NOT NULL
...             AND month IS NOT NULL
...             AND time IS NOT NULL
...             AND caller IS NOT NULL
...             AND called IS NOT NULL
...           PRIMARY KEY((called, year), month, time, caller);
```

- Anrufe werden hier nach Angerufenen und Jahr partitioniert
- Die View wird von Cassandra selbst aktuell gehalten
- Bestehende Primärschlüssel-Felder müssen in einer Materialized View mindestens im Clustering Key sein

- **Lightweight Transactions** (LWT) erlauben schnelle Compare-and-Swap-Operationen (CAS)

```
cqlsh:jugsaar> UPDATE call_detail_record SET duration_seconds=927
... WHERE year=2022 AND month=12 AND caller='+49160123987456' AND time='2022-12-30 19:00:00'
... IF duration_seconds=634;

[applied]
-----
True
```

- Die Änderung (swap) wird nur durchgeführt, wenn die Voraussetzungen noch so sind wie erwartet (compare)
- So werden inkrementelle Änderungen, z.B. Berechnungen auf bestehenden Werten, idempotent

- **Counter** sind Ganzzahlfelder mit eingebauter CAS-Fähigkeit

```
cqlsh:jugsaar> CREATE TABLE call_count (  
    ...     year    INT,  
    ...     month   INT,  
    ...     caller  TEXT,  
    ...     calls   COUNTER,  
    ...     PRIMARY KEY ( year, month, caller )  
    ... );
```

- Counter dürfen nur in **Counter Tables** vorkommen und Counter Tables dürfen nur Counter-Felder enthalten
- Counter können nicht gesetzt, sondern nur hoch- und runtergezählt werden:

```
cqlsh:jugsaar> UPDATE call_count SET calls=calls+1  
    ... WHERE year=2022 AND month=12 AND caller='+49160123987456';  
cqlsh:jugsaar> SELECT * FROM call_count;
```

year	month	caller	calls
2022	12	+49160123987456	1

(1 rows)

- Batches erlauben **atomare Transaktionen**

```
cqlsh:jugsaar> BEGIN BATCH
... INSERT INTO call_detail_record (year, month, caller, time, called, duration_seconds)
... VALUES (2022, 12, '+49151871368765', '2022-12-31 18:00:00', '+49172836583298', 105);
... INSERT INTO call_detail_record (year, month, caller, time, called, duration_seconds)
... VALUES (2022, 11, '+49152748360123', '2022-11-15 18:30:00', '+49170194499273', 457);
... APPLY BATCH;
```

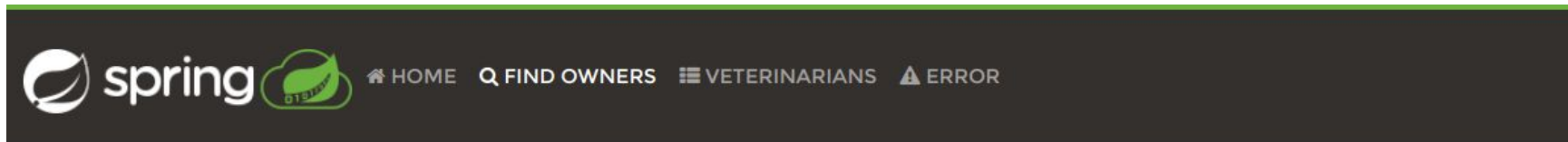
- **Batch** = mehrere INSERT, UPDATE und DELETE auf mehreren Tabellen und Partitionen
 - **atomare Transaktion** = die Änderungen werden entweder komplett oder gar nicht ausgeführt
 - innerhalb einer Partition sind Änderungen auch **isoliert**, d.h. werden erst nach Abschluss aller Änderungen sichtbar
 - verschiedene Partitionen können aber zu unterschiedlichen Zeiten fertig und sichtbar werden, daher keine vollständige Isolation
- Performance
 - LOGGED Batches sind der Standard: immer atomar, aber vergleichsweise langsam
 - UNLOGGED Batches sind eine schnellere Alternative, wenn es ok ist, dass auch nur ein Teil der Partitionen aktualisiert wird



IV Entwicklung

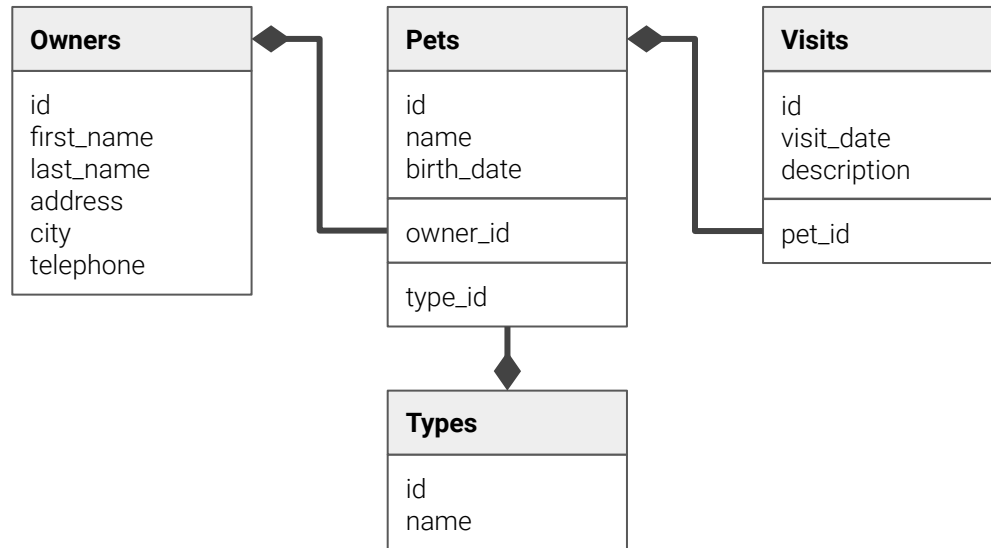
—
Datenmodellierung und Code

- Idealtypische Demo-Webanwendung zum Spring Framework
 - Open Source, <https://spring-petclinic.github.io/>
 - Inzwischen viele Forks mit anderen Technologien unter <https://spring-petclinic.github.io/docs/forks.html>



Owners

Name	Address	City	Telephone	Pets
George Franklin	110 W. Liberty St.	Madison	6085551023	Leo
Betty Davis	638 Cardinal Ave.	Sun Prairie	6085551749	Basil
Eduardo Rodriguez	2693 Commerce St.	McFarland	6085558763	Jewel, Rosy
Harold Davis	563 Friendly St.	Windsor	6085553198	Iggy
Peter McTavish	2387 S. Fair Way	Madison	6085552765	George



“Data First” Design

1. Entities und Relationen identifizieren
2. Normalisierte Tabellen erstellen
3. Queries nach Bedarf bauen

■ Klassisches relationales Datenbankschema

- Ein Besitzer hat beliebig viele Haustiere verschiedener Art
- Jedes Haustier hat beliebig viele Arzttermine
- Jeder Arzt ist auf verschiedene Themen spezialisiert

- <https://github.com/spring-projects/spring-petclinic/blob/main/src/main/resources/db/postgres/schema.sql>

1. Was muss die Anwendung können?

- Besitzer nach Namen suchen, anlegen, ändern
- Haustiere eines Besitzers auflisten, anlegen, ändern
- Termine eines Haustiers auflisten, anlegen, ändern
- Arzt nach Spezialisierung suchen
- Ärzte und ihre Spezialisierungen auflisten, anlegen, ändern

“Application First” Design

1. Anwendung skizzieren
2. Queries und Tabellen identifizieren
3. Batches zusammenstellen

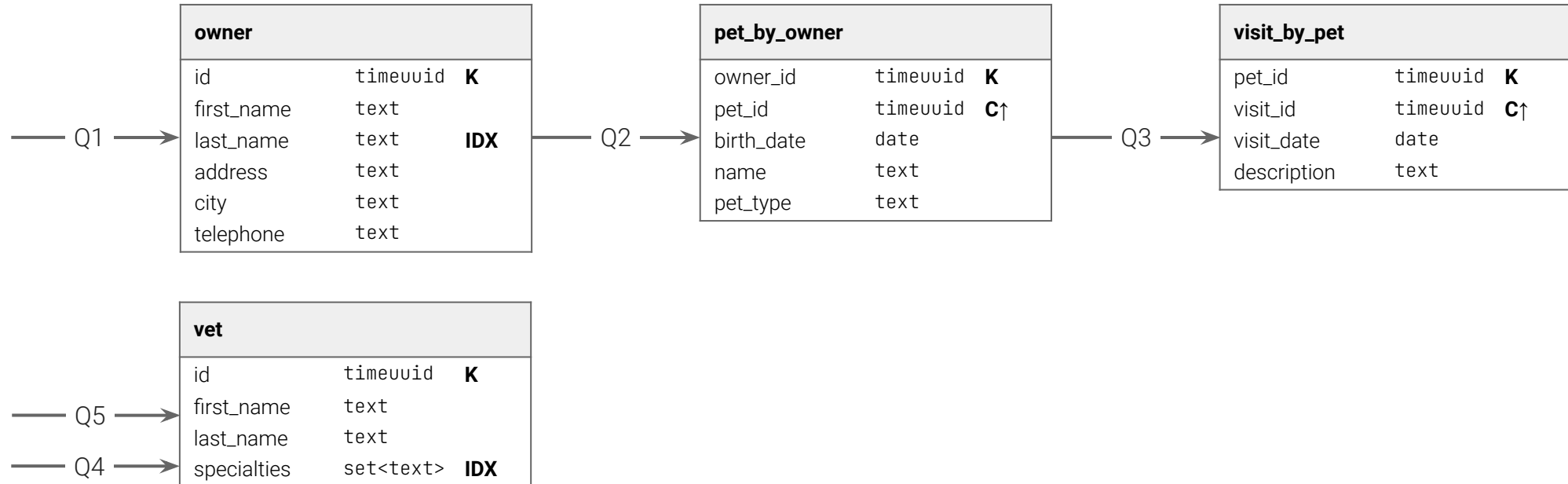


Owners

Name	Address	City	Telephone	Pets
George Franklin	110 W. Liberty St.	Madison	6085551023	Leo
Betty Davis	638 Cardinal Ave.	Sun Prairie	6085551749	Basil
Eduardo Rodriguez	2693 Commerce St.	McFarland	6085558763	Jewel, Rosy
Harold Davis	563 Friendly St.	Windsor	6085553198	Iggy
Peter McTavish	2387 S. Fair Way	Madison	6085552765	George

2. Queries und Tabellen als **Chebotko-Diagramm**

- Q1: Besitzer nach Namen suchen, anlegen, ändern
- Q2: Haustiere eines Besitzers auflisten, anlegen, ändern
- Q3: Termine eines Haustiers auflisten, anlegen, ändern
- Q4: Arzt nach Spezialisierung suchen
- Q5: Ärzte und ihre Spezialisierungen auflisten, anlegen, ändern



2. Queries und Tabellen

- Q1: Besitzer nach Namen suchen, anlegen, ändern

- Schema

```
CREATE TABLE owner (  
    id            timeuuid,  
    first_name    text,  
    last_name     text,  
    address       text,  
    city          text,  
    telephone     text,  
    PRIMARY KEY (id)  
);  
  
CREATE CUSTOM INDEX owner_name ON owner (last_name) USING 'org.apache.cassandra.index.sasi.SASIIndex'  
WITH OPTIONS = {  
    'mode': 'CONTAINS', 'analyzer_class': 'org.apache.cassandra.index.sasi.analyzer.StandardAnalyzer',  
    'tokenization_normalize_lowercase': 'true', 'tokenization_locale': 'de'  
};
```

- Queries

```
SELECT * FROM owner WHERE last_name LIKE '%dav%';
```

(UPDATE und DELETE sind trivial, daher ohne Beispiele)

2. Queries und Tabellen

- Q2: Haustiere eines Besitzers auflisten, anlegen, ändern

- Schema

```
CREATE TABLE pet_by_owner (  
  owner_id  timeuuid,  
  pet_id    timeuuid,  
  birth_date date,  
  name      text,  
  pet_type  text,  
  PRIMARY KEY (owner_id, pet_id)  
);
```

- Queries

```
SELECT * FROM pet_by_owner WHERE owner_id=da1dc18e-8a1b-11ed-ac3f-14857f0c9b17;
```

(UPDATE und DELETE sind trivial, daher ohne Beispiele)

2. Queries und Tabellen

- Q3: Termine eines Haustiers auflisten, anlegen, ändern

- Schema

```
CREATE TABLE visit_by_pet (  
    pet_id      timeuuid,  
    visit_id    timeuuid,  
    visit_date  date,  
    description text,  
    PRIMARY KEY (pet_id, visit_id)  
);
```

- Queries

```
SELECT * FROM visit_by_pet WHERE pet_id=81877cda-8a1c-11ed-896b-14857f0c9b17;
```

(UPDATE und DELETE sind trivial, daher ohne Beispiele)

2. Queries und Tabellen

- Q4: Arzt nach Spezialisierung suchen
- Q5: Ärzte und ihre Spezialisierungen auflisten, anlegen, ändern

- Schema

```
CREATE TABLE vet (  
    id            timeuuid,  
    first_name    text,  
    last_name     text,  
    specialties   set<text>,  
    PRIMARY KEY (id)  
);  
CREATE INDEX ON vet(specialties);
```

- Queries

```
SELECT * FROM vet WHERE specialties CONTAINS 'surgery';
```

(SELECT ohne Bedingungen, UPDATE und DELETE sind trivial, daher ohne Beispiele)

3. Batches zusammenstellen

- Besitzerwechsel: Iggy, der Leguan, zieht von Harold Davis zu David Schroeder

```
-- Daten von Iggy auslesen
SELECT * FROM pet_by_owner
WHERE owner_id=d84f2e7e-8a1b-11ed-9e39-14857f0c9b17 AND pet_id=80ed39b8-8a1c-11ed-a6fd-14857f0c9b17;

BEGIN BATCH

-- Iggy bei David Schroeder eintragen
INSERT INTO pet_by_owner (owner_id, pet_id, birth_date, name, pet_type)
VALUES (d9d0bf24-8a1b-11ed-a68e-14857f0c9b17, 80ed39b8-8a1c-11ed-a6fd-14857f0c9b17, '2000-11-30', 'Iggy', 'lizard');

-- Iggy bei Harold Davis löschen
DELETE FROM pet_by_owner WHERE owner_id=d84f2e7e-8a1b-11ed-9e39-14857f0c9b17 AND pet_id=80ed39b8-8a1c-11ed-a6fd-14857f0c9b17;

APPLY BATCH;
```

- Sonst noch Änderungen, die Batches erfordern könnten?

Was	Technisches Maximum	Empfohlenes Maximum
Schlüssel-Wert-Paare pro Partition	ca. 2 Mrd. (2^{31})	100.000
Gesamtgröße einer Partition auf Festplatte	-	100 MB, optimal sind 10 MB
Größe eines einzelnen Werts oder Blobs	2 GB	1 MB
Größe eines Werts im Clustering Key	65.535 Bytes ($2^{16} - 1$)	
Größe eines Partition Key	65.535 Bytes ($2^{16} - 1$)	
Länge eines Tabellen- oder Keyspace-Namens	48 Zeichen	
Anzahl Parameter in einer Query	65.535 ($2^{16} - 1$)	
Anzahl Änderungen pro Batch	65.535 ($2^{16} - 1$)	
Felder in einem Tupel	32.768 (2^{15})	10
Anzahl Elemente pro List, Map oder Set	ca. 2 Mrd. (2^{31})	
Größe eines Werts in einer List, Map oder Set	65.535 Bytes ($2^{16} - 1$)	
Anzahl Keys in einer Map	65.535 ($2^{16} - 1$)	

- Zurück zum Anfang: unser Schema für den Einzelverbindungs-nachweis hat ein Problem!
 - Warum würde das Schema z.B. bei Vodafone nicht funktionieren?
 - Welche einfache Änderung könnte das Problem beheben?

```
cqlsh:jugsaar> SELECT * FROM call_detail_record WHERE year=2022 AND month=12;
```

year	month	caller	time	called	duration_seconds
2022	12	+49151746328	2022-12-24 18:10:00.000000+0000	+49172836492	112
2022	12	+49160123987	2022-12-09 10:00:00.000000+0000	+49172456987	634
2022	12	+49160123987	2022-12-30 08:42:00.000000+0000	+49154828374	89

- Lösung
 - Die Partitionen werden viel zu groß, wenn alle Gespräche aller Kunden eines Monats in nur einer Partition landen.
 - Man braucht für einen Nachweis immer nur die Daten eines Kunden in einem Monat, also:

```
cqlsh:jugsaar> SELECT * FROM call_detail_record WHERE year=2022 AND month=12 AND caller='+49160123987';
```

year	month	caller	time	called	duration_seconds
2022	12	+49160123987	2022-12-09 10:00:00.000000+0000	+49172456987	634
2022	12	+49160123987	2022-12-30 08:42:00.000000+0000	+49154828374	89

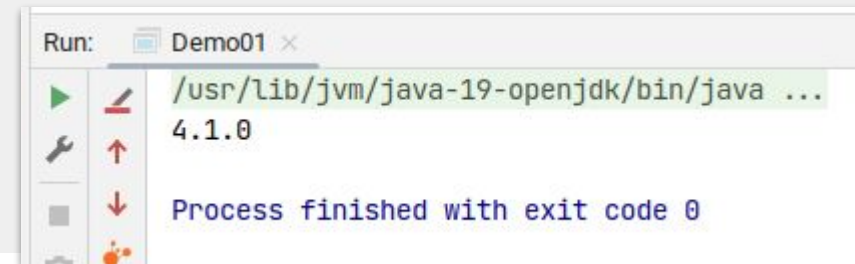
1. Treiber einbinden via Maven

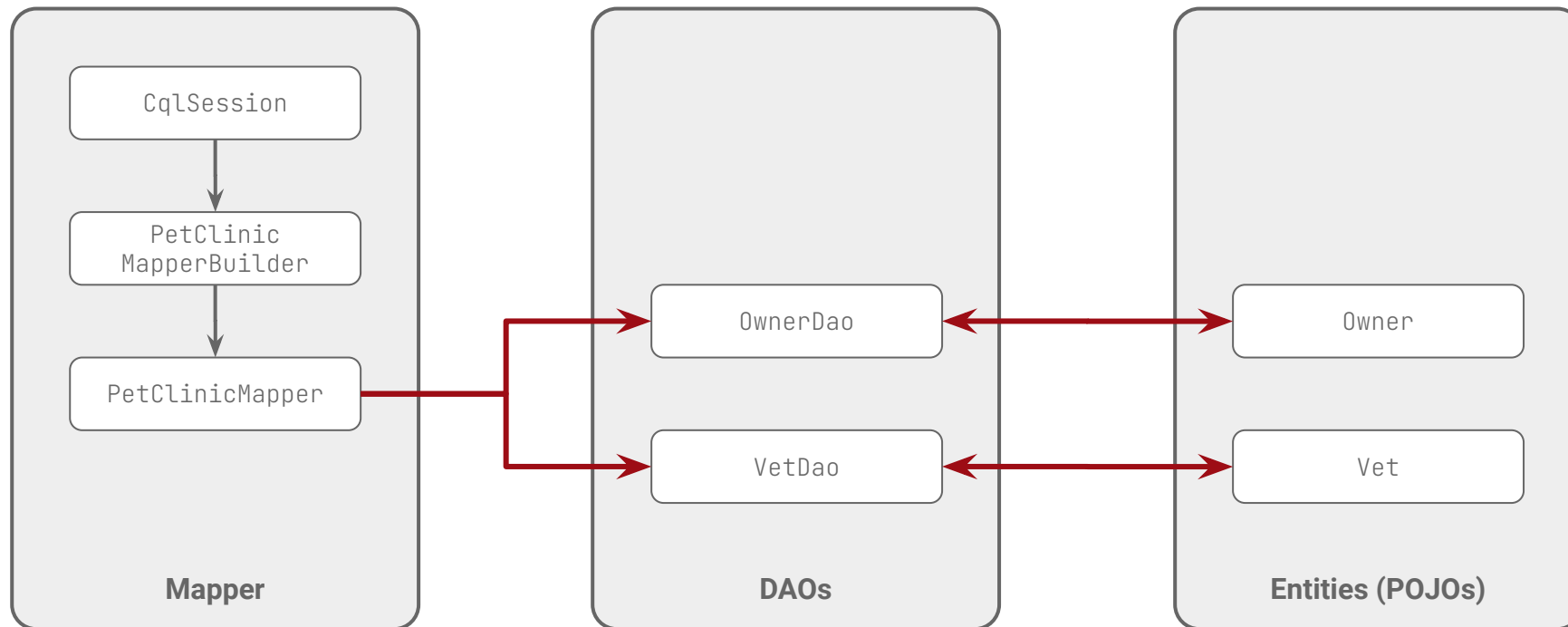
```
<dependencies>
  <dependency>
    <groupId>com.datastax.oss</groupId>
    <artifactId>java-driver-core</artifactId>
    <version>4.14.1</version>
  </dependency>
</dependencies>
```

2. Session, Query, ResultSet, Row, fertig

```
public class Demo01 {

    public static void main(String[] args) {
        try (CqlSession session = CqlSession.builder()
            .addContactPoint(new InetSocketAddress("127.0.0.1", 9042))
            .build()) {
            ResultSet rs = session.execute("SELECT release_version FROM system.local");
            Row row = rs.one();
            System.out.println(row.getString("release_version"));
        }
    }
}
```





1. Annotation Processors

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.10.1</version>
      <configuration>
        <release>19</release>
        <compilerArgs>--enable-preview</compilerArgs>
        <annotationProcessorPaths>
          <path>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>${lombok.version}</version>
          </path>
          <path>
            <groupId>com.datastax.oss</groupId>
            <artifactId>java-driver-mapper-processor</artifactId>
            <version>${datastax-driver.version}</version>
          </path>
        </annotationProcessorPaths>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Annotation Processors:

erst Lombok,
dann DataStax!

2. Dependencies

```
<dependencies>
  <dependency>
    <groupId>com.datastax.oss</groupId>
    <artifactId>java-driver-core</artifactId>
    <version>${datastax-driver.version}</version>
  </dependency>
  <dependency>
    <groupId>com.datastax.oss</groupId>
    <artifactId>java-driver-mapper-runtime</artifactId>
    <version>${datastax-driver.version}</version>
  </dependency>
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>${lombok.version}</version>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-nop</artifactId>
    <version>1.7.36</version>
  </dependency>
</dependencies>
```

3. Entity

```
import com.datastax.oss.driver.api.mapper.annotations.Entity;
import com.datastax.oss.driver.api.mapper.annotations.PartitionKey;
import lombok.Data;

import java.util.UUID;

@Data
@Entity
public class Owner {

    @PartitionKey private UUID id;
    private String firstName;
    private String lastName;
    private String address;
    private String city;
    private String telephone;
}
```

4. DAO

```
import com.datastax.oss.driver.api.core.PagingIterable;
import com.datastax.oss.driver.api.mapper.annotations.Dao;
import com.datastax.oss.driver.api.mapper.annotations.Delete;
import com.datastax.oss.driver.api.mapper.annotations.Insert;
import com.datastax.oss.driver.api.mapper.annotations.Select;

import java.util.UUID;

@Dao
public interface OwnerDao {

    @Select(customWhereClause = "last_name LIKE :searchString")
    PagingIterable<Owner> findByName(String searchString);

    @Select
    Owner findById(UUID ownerId);

    @Insert
    void save(Owner owner);

    @Delete
    void delete(Owner owner);
}
```

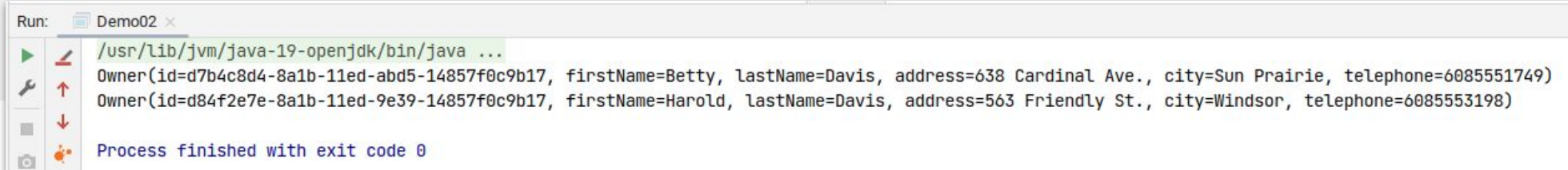
4. Mapper definieren

```
@Mapper
public interface PetClinicMapper {
    @DaoFactory
    OwnerDao ownerDao(@DaoKeyspace CqlIdentifier keyspace);
}
```

5. Mapper erzeugen lassen, DAO besorgen, Abfragen machen

```
public class Demo02 {

    public static void main(String[] args) {
        try (CqlSession session = CqlSession.builder().build()) {
            final PetClinicMapper mapper = new PetClinicMapperBuilder(session).build();
            final OwnerDao ownerDao = mapper.ownerDao(CqlIdentifier.fromCql("petclinic"));
            final PagingIterable<Owner> owners = ownerDao.findByName("%dav%");
            owners.forEach(System.out::println);
        }
    }
}
```



```
Run: Demo02 x
/usr/lib/jvm/java-19-openjdk/bin/java ...
Owner(id=d7b4c8d4-8a1b-11ed-abd5-14857f0c9b17, firstName=Betty, lastName=Davis, address=638 Cardinal Ave., city=Sun Prairie, telephone=6085551749)
Owner(id=d84f2e7e-8a1b-11ed-9e39-14857f0c9b17, firstName=Harold, lastName=Davis, address=563 Friendly St., city=Windsor, telephone=6085553198)
Process finished with exit code 0
```



V Operations

Cassandra produktiv
betreiben

- Serverhardware für Produktivbetrieb
 - min. 2 CPUs und 8 GB RAM (ernsthafte Setups starten mit 8 CPUs und 32 GB RAM) und mind. Gigabit-Netzwerk
 - Genug Festplattenplatz unter `/var/lib/cassandra` (ext4 ist am schnellsten)
 - Commit Logs und Daten kann man getrennt speichern, lohnt sich bei modernen SSDs aber kaum noch
 - Cassandra repliziert Daten selbst, daher kein NFS, SAN, RAID1+, ...
 - Analog: Cloud-VMs mit lokalen "flüchtigen" NVMe
- Software
 - Gute Erfahrungen mit Debian 11 und OpenJDK 11

```
echo "deb https://debian.cassandra.apache.org 41x main" > /etc/apt/sources.list.d/cassandra.sources.list
wget -O /etc/apt/trusted.gpg.d/cassandra.asc https://downloads.apache.org/cassandra/KEYS
apt-get update
systemctl mask cassandra.service # verhindert automatischen Start bei der Installation!
apt-get install -y --no-install-recommends openjdk-11-jre-headless cassandra cassandra-tools

# Dann Konfiguration in /etc/cassandra anpassen, mind. cassandra.yaml und cassandra-rackdc.properties

systemctl unmask cassandra.service
systemctl enable cassandra.service
systemctl start cassandra
tail -f /var/log/cassandra/system.log # warten auf "state jump to NORMAL" und "Startup complete"
```

- System
 - Swap und Defragmentierung von Huge Pages abschalten
 - **Feste IPs** und ggf. Hostnamen vergeben
 - **Zeitsynchronisation mit NTP** aktivieren
 - TCP-Kernel-Parameter mit `sysctl` anpassen
 - `ulimits` für den `cassandra`-User setzt das Installationsskript schon rauf
- Cassandra-Config in `/etc/cassandra`
 - `cassandra-rackdc.properties`: Data Center und Rack des Knotens
 - `cassandra.yaml`: **cluster_name**, **seeds**, **listen_address**, **rpc_address**, `num_tokens` sowie ggf. Verschlüsselung (`server_encryption_options`, `client_encryption_options`, `transparent_data_encryption_options`)
 - `jvm11-options.conf`: G1-Garbage-Collector aktivieren

https://cassandra.apache.org/doc/latest/cassandra/getting_started/production.html

https://docs.datastax.com/en/dse/6.8/dse-admin/datastax_enterprise/config/configRecommendedSettings.html

<https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/operations/opsTuneJVM.html>

https://raw.githubusercontent.com/jolynch/performance-analysis/master/notebooks/cassandra_availability/whitepaper/cassandra-availability-virtual.pdf

<https://cassandra.apache.org/doc/latest/cassandra/operating/security.html>

Aufgabe	Befehl
Hilfe zu jedem Befehl	<code>nodetool help <i>befehl</i></code>
Clusterzustand	<code>nodetool status [<i>keyspace</i>]</code>
Verteilung der Partitionsgrößen einer Tabelle	<code>nodetool tablehistograms <i>keypace table</i></code>
Detaillierte Statistik zu einer Tabelle	<code>nodetool tablestats -H <i>keyspace.table</i></code>
Wo ist gerade am meisten los?	<code>nodetool toppartitions</code>
Last durch den Garbage Collector	<code>nodetool gcstats</code>
Auslastung der Thread Pools	<code>nodetool tpstats</code>
Netzwerkauslastung	<code>nodetool netstats</code>
Clientverbindungen zum Cluster	<code>nodetool clientstats</code>

Aufgabe	Befehl
Snapshot aller Keyspaces anlegen	<code>nodetool snapshot</code>
Snapshot eines Keyspace anlegen	<code>nodetool snapshot keypace</code>
Snapshots auflisten	<code>nodetool listsnapshots</code>
Snapshot löschen	<code>nodetool clearsnapshot -t name</code>
Incremental Backups aktivieren	<code>nodetool enablebackup</code>
Snapshot zurückspielen <ul style="list-style-type: none">- um Daten komplett zu ersetzen, vorher <code>TRUNCATE table</code>- Schema ggf. aus <code>snapshots/name/schema.cql</code> wiederherstellen	<pre># Auf allen Knoten für alle betroffenen SSTables: cd /var/lib/cassandra/data/keyspace/sstable cp snapshots/name/* . nodetool refresh systemctl restart cassandra</pre>

Aufgabe**Befehl**

Compaction einer oder mehrerer Tabellen auf dem angespr. Knoten starten

```
nodetool compact keypace table table ...
```

Laufende Compactions auf dem angesprochenen Knoten anzeigen

```
nodetool compactionstats
```

Compaction auf dem angesprochenen Knoten stoppen

```
nodetool stop -id compaction_id
```

Abgeschlossene Compactions auf dem angesprochenen Knoten zeigen

```
nodetool compactionhistory
```

Compaction Strategy einer Tabelle im Cluster ändern

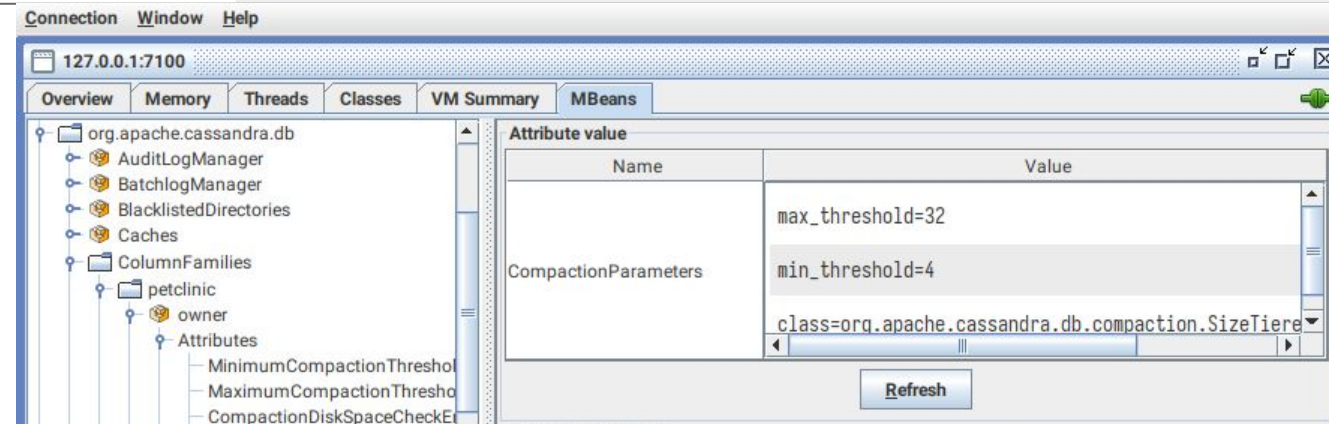
```
cqlsh> ALTER TABLE table WITH  
... compaction = {'class': 'LeveledCompactionStrategy'};
```

- *Achtung: erzeugt auf allen Knoten gleichzeitig hohe Last durch die Compaction aller Daten*

"Schweizer Java-Messer"

```
nodetool sjk --help
```

JConsole



Aufgabe	Befehl
<i>Bei allen Repair-Befehlen werden immer nur die Replicas der Daten repariert, die dem angesprochenen Knoten gehören, nicht die ganze Tabelle überall!</i>	
Incremental Repair über alle Replicas gleichzeitig (parallel) starten - überspringt als bereits repariert markierte Partitionen	<code>nodetool repair [keypace [table...]]</code>
Full Repair aller Replicas gleichzeitig (parallel) starten - repariert alle Daten unabhängig von Incremental-Repair-Markierung - hilft nach Admin-Fehlern, Schäden an SSTables etc.	<code>nodetool repair -full [keypace [table...]]</code>
Full Repair aller Replicas nacheinander (sequenziell) starten - dauert länger, macht aber weniger Last auf einmal	<code>nodetool repair -full -seq [keypace [table...]]</code>
Full Repair nur von Replicas im angegebenen Data Center	<code>nodetool repair -full -dc datacenter [keypace [table...]]</code>
Full Repair nur von Replicas im selben Data Center wie der angesprochene Knoten	<code>nodetool repair -full -local [keypace [table...]]</code>
Full Repair in allen Data Centers parallel, pro Data Center aber nacheinander	<code>nodetool repair -full -dcpair [keypace [table...]]</code>
Laufende Repairs anzeigen	<code>nodetool repair_admin</code>
Incremental Repair abbrechen	<code>nodetool repair_admin cancel --session id</code>

Aufgabe	Befehl
Daten auf dem Knoten aus den Replicas wiederherstellen	<code>nodetool rebuild</code>
Secondary Index neu aufbauen	<code>nodetool rebuild_index <i>keypace table index</i></code>
Daten, die nicht auf den Knoten gehören, entfernen	<code>nodetool cleanup</code>
Gelöschte Daten (> 10 Tage alte Tombstones) aus einer Tabelle entfernen	<code>nodetool garbagecollect <i>keypace table</i></code>
Aufbau einer Materialized View überwachen	<code>nodetool viewbuildstatus <i>keypace view</i></code>

Aufgabe	Befehl
Daten aus beschädigten SSTables wiederherstellen	<code>nodetool scrub <i>keypace table</i></code>
SSTables nach Änderungen auf Festplatte neu einlesen	<code>nodetool refresh</code>
SSTables nach Cassandra-Update auf neue Version aktualisieren	<code>nodetool upgradesstables</code>

Aufgabe	Befehl
Daten aus den Memtables auf Platte schreiben	<code>nodetool flush</code>
Knoten deaktivieren und Daten auf Platte schreiben	<code>nodetool drain</code>
Knoten kontrolliert außer Betrieb nehmen	<code>nodetool decommission</code>
Inaktiven Knoten aus dem Cluster entfernen	<code>nodetool removemode host_id</code>
Defekten Knoten gewaltsam aus dem Cluster entfernen	<code>nodetool assassinate ip_address</code>
Neuen Knoten im Cluster aufnehmen - danach <i>cleanup</i> auf übrigen Knoten ausführen	<code>nodetool join</code>
Tokenverteilung anzeigen	<code>nodetool ring</code>
Tokenverteilung eines Keyspace anzeigen	<code>nodetool describering keyspace</code>
Knoten im Ring verschieben	<code>nodetool move -- token</code>

■ Minor Upgrades

- Knotenweises Update und Reboot reicht:

```
sudo apt update
sudo apt upgrade
sudo reboot
```

- Komfortabler per Automatisierung, z.B. via Ansible →

■ Major Release

- Backup
- `nodetool drain`
- Cassandra anhalten
- Linux, glibc, Kernel, Java, Cassandra aktualisieren
- Reboot, Cassandra startet dann automatisch
- `/var/log/cassandra/system.log` prüfen
- `nodetool upgradesstables`

```
- name: Update and reboot Cassandra Cluster JUGSaar
  hosts: jugsaar
  become: yes
  serial: 1
  tasks:

- name: Upgrade packages (apt)
  apt:
    upgrade: safe
    update_cache: yes

- name: Reboot and wait
  reboot:

- name: Wait for restart to complete
  wait_for:
    host: "{{ansible_default_ipv4.address}}"
    port: 9042
```



Erste Schritte in Cassandra

mit CCM




```
~ virtualenv --python=python3 --clear --always-copy cassandra-venv
created virtual environment CPython3.10.9.final.0-64 in 142ms
```

```
...
```

```
~ pip install ccm
```

```
...
```

```
Successfully installed ccm-3.1.5 pyYaml-6.0 six-1.16.0
```

```
~ java -version
```

```
openjdk version "11.0.17" 2022-10-18
```

```
...
```

Wichtig: Java 11 und JAVA_HOME!

```
~ ccm create -v 4.1.0 -n 3 jugsaar
```

```
Current cluster is now: jugsaar
```

Kopiert Cassandra nach ~/.ccm/repository/<version>

```
~ ccm start
```

```
[node1 ERROR] b'OpenJDK 64-Bit Server VM warning: Option UseConcMarkSweepGC was deprecated in version 9.0 and will likely be removed in a future release.'
```

```
...
```

```
~ ccm status
```

```
Cluster: 'jugsaar'
```

```
-----
```

```
node1: UP
```

```
node2: UP
```

```
node3: UP
```

node1	127.0.0.1
-------	-----------

node2	127.0.0.2
-------	-----------

node3	127.0.0.3
-------	-----------



Pull requests



riptano / ccm

Public

<> Code

Issues 61

Pull requests 9

Actions

```
~ ccm stop
```

```
~ sed -i.orig '/sasi_indexes_enabled/c sasi_indexes_enabled: true' ~/.ccm/jugsaar/node*/conf/cassandra.yaml
```

```
~ ccm start --wait-for-binary-proto
```

```
[node1 ERROR] b'OpenJDK 64-Bit Server VM warning: Option UseConcM  
removed in a future release.'
```

```
'''
```

```
~ export PATH="$PATH:~/.ccm/repository/4.1.0/bin"
```

```
~ cqlsh
```

```
Connected to jugsaar at 127.0.0.1:9042
```

```
[cqlsh 6.1.0 | Cassandra 4.1.0 | CQL spec 3.4.6 | Native protocol v5]
```

```
Use HELP for help.
```

```
cqlsh> source 'petclinic.cql';
```

```
Warnings :
```

```
SASI indexes are experimental and are not recommended for production use.
```

```
cqlsh> use petclinic;
```

```
cqlsh:petclinic> desc tables;
```

```
owner  pet_by_owner  vet  visit_by_pet
```

```
cqlsh:petclinic> █
```

Config der Clusterknoten liegt unter
~/.ccm/cluster_name/node*/conf und
kann ganz normal geändert werden

will likely be

```
~ cd .ccm/jugsaar/node1/data0/petclinic
```

```
~/ccm/jugsaar/node1/data0/petclinic ls
```

```
owner-db16ac108aa011eda7c691f9b464fc64      vet-dd94a4608aa011eda7c691f9b464fc64
pet_by_owner-dc5d35d08aa011eda7c691f9b464fc64  visit_by_pet-dcea81b08aa011eda7c691f9b464fc64
```

```
~/ccm/jugsaar/node1/data0/petclinic ls -l owner-db16ac108aa011eda7c691f9b464fc64
```

```
total 0
drwxr-xr-x 1 pwalter pwalter 0  2. Jan 14:24 backups
```

```
~/ccm/jugsaar/node1/data0/petclinic source ~/cassandra-venv/bin/activate
```

```
~/ccm/jugsaar/node1/data0/petclinic ccm node1 flush
```

```
...
```

```
~/ccm/jugsaar/node1/data0/petclinic ls -l owner-db16ac108aa011eda7c691f9b464fc64
```

```
total 52
drwxr-xr-x 1 pwalter pwalter      0  2. Jan 14:24 backups
-rw-r--r-- 1 pwalter pwalter    47  2. Jan 14:49 nb-1-big-CompressionInfo.db
-rw-r--r-- 1 pwalter pwalter   228  2. Jan 14:49 nb-1-big-Data.db
-rw-r--r-- 1 pwalter pwalter     9  2. Jan 14:49 nb-1-big-Digest.crc32
-rw-r--r-- 1 pwalter pwalter    16  2. Jan 14:49 nb-1-big-Filter.db
-rw-r--r-- 1 pwalter pwalter    61  2. Jan 14:49 nb-1-big-Index.db
-rw-r--r-- 1 pwalter pwalter  12312  2. Jan 14:49 nb-1-big-SI_owner_name.db
-rw-r--r-- 1 pwalter pwalter   4927  2. Jan 14:49 nb-1-big-Statistics.db
-rw-r--r-- 1 pwalter pwalter    92  2. Jan 14:49 nb-1-big-Summary.db
-rw-r--r-- 1 pwalter pwalter   109  2. Jan 14:49 nb-1-big-TOC.txt
```

Datacenter: datacenter1

Status=Up/Down

```
|/ State=Normal/Leaving/Joining/Moving
```

--	Address	Load	Tokens	Owns (effective)
----	---------	------	--------	------------------

UN 127.0.0.2 104.92 KiB 1 33.3%

```
UN 127.0.0.1 179.12 KiB 1 33.3%
```

```
UN 127.0.0.3 104.93 KiB 1 33.3%
```

```
node1: UP
```

```
cluster=jugsaar
```

```
auto_bootstrap=False
```

```
binary=('127.0.0.1', 9042)
```

```
storage=('127.0.0.1', 7000)
```

jmx_port=7100

```
remote_debug_port=0
```

```
byteman_port=0
```

```
initial_token=-9223372036854775808
```

pid=692753

```
node1      127.0.0.1      7100
```

```
node2      127.0.0.1      7200
```

```
node3      127.0.0.1      7300
```

	Rack
i3f01c8f49d2	rack1
-3ed5b7f02c67	rack1
-81a18e2979e3	rack1

- Literatur
 - Carpenter/Hewitt (2020): Cassandra - The Definitive Guide
 - Petrov (2019): Database Internals
- Online-Quellen und -Kurse
 - Offizielle Website: <https://cassandra.apache.org/>
 - ↳ Quellcode: <https://github.com/apache/cassandra>
 - DataStax for Developers: <https://www.datastax.com/dev>
 - Awesome Cassandra: <https://github.com/Anant/awesome-cassandra>
- Was nicht in diesen Vortrag gepasst hat
 - Tuning und Lasttests mit `cassandra_stress`
 - ↳ <https://www.sestevez.com/sestevez/CassandraDataModeler/>
 - Backups
 - ↳ <https://github.com/instacluster/esop>
 - ↳ <https://github.com/thelastpickle/cassandra-medusa>
 - Integration mit Kafka, Spark, Pig, Hive, ScalarDB, Cadence, ...

