Big Data – Challenges of Distributed Data-Systems: Partitioning

Winter Semester 2025/2026,
Cooperative State University Baden-Wuerttemberg

Agenda – 13.10.2025

01

Presentation and Discussion: Exercise Of Last Lecture

Write, compile and run Java MapReduce,
Hive, HiveQL and external tables

02

Introduction To The Challenges Of Distributed Data-Systems: Partitioning

Basics of Partitioning, Key-Range and Hash Partitioning, Partitioning of Secondary Indices, Rebalancing and Lookup of Partitions

03

HandsOn – Hive, HiveQL, Hive/HDFS Partitioning and HiveServer2

Quick Introduction To Hive, HiveQL, Hive/HDFS Partitioning and HiveQL via JDBC (HiveServer2)

04

Exercise – HDFS/Hive work with Partitions on IMDb dataset.

HiveServer2, HiveQL via JDBC, Partitioning with HDFS and Hive

Schedule

Date	Time	Room	Lecture Topic	HandsOn
29.09.2025	13:15-15:45	Ro. N/A	About This Lecture, Introduction to Big Data	Setup Google Cloud, Create Own Hadoop Cluster and Run MapReduce
06.10.2025	13:15-15:45	Ro. N/A	(Non-)Functional Requirements Of Distributed Data-Systems, Data Models and Access	Hive and HiveQL
13.10.2025	**13:15-15:45**	**Ro. N/A**	**Challenges Of Distributed Data Systems: Partitioning**	HiveQL via JDBC, Data Partitioning (with HDFS and Hive)
20.10.2025	13:15-15:45	Ro. N/A	Challenges Of Distributed Data Systems: Replication	Spark and Scala
27.10.2025	13:15-15:45	Ro. N/A	ETL Workflow and Automation	PySpark and Notebooks (Jupyter)
03.11.2025	13:15-15:45	Ro. N/A	Batch and Stream Processing	Airflow
10.11.2025	13:15-15:45	Ro. N/A	Practical Exam	Work On Practical Exam
17.11.2025	13:15-15:45	Ro. N/A	Practical Exam	Work On Practical Exam
24.11.2025	**13:15-15:45**	**Ro. N/A**	**Practical Exam Presentation**	
01.12.2025	**13:15-15:45**	**Ro. N/A**	**Practical Exam Presentation**	

Solution – Exercise II

HiveQL, Create and work with External Tables on IMDb Data

Solution

Prerequisites:

- Setup Google Cloud SDK
- Start VM instance
- Pull docker container `marcelmittelstaedt/hive_base:latest`
- Start docker container: `docker run -dit --name hive_base_container -p 8088:8088 -p 9870:9870 -p 9864:9864 marcelmittelstaedt/hive_base:latest`
- Get into docker container
- Start Hadoop and Hive Shell:
    - `start-all.sh`
    - `hive`

Solution

Exercise 1-4:

1. Download and unzip https://datasets.imdbws.com/name.basics.tsv.gz

```
wget https://datasets.imdbws.com/name.basics.tsv.gz
gunzip name.basics.tsv.gz
```

2. Create HDFS directory **/user/hadoop/imdb/name_basics/** for file name.basics.tsv

```
hadoop fs -mkdir /user/hadoop/imdb/name_basics
```

3. Put TSV file to HDFS:

```
hadoop fs -put name.basics.tsv /user/hadoop/imdb/name_basics/name.basics.tsv
```

Solution

Exercise 1-4:

4. Create Hive Table **name_basics**:

```
hive > CREATE EXTERNAL TABLE IF NOT EXISTS name_basics(
    nconst STRING,
    primary_name STRING,
    birth_year INT,
    death_year STRING,
    primary_profession STRING,
    known_for_titles STRING
    ) COMMENT 'IMDb Actors' ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' ST
ORED AS TEXTFILE LOCATION '/user/hadoop/imdb/name_basics'
TBLPROPERTIES ('skip.header.line.count'='1');
```

Solution

Exercise 5:

a) How many movies and how many TV series are within the IMDB dataset?

```
hive >   SELECT m.title_type, count(*)
         FROM title_basics m GROUP BY m.title_type;

tvMovie 152520
movie 727439
tvEpisode 9200239
tvSeries 288488
[…]

Time taken: 32.908 seconds, Fetched: 11 row(s)
```

b) Who is the youngest actor/writer/… within the dataset?

```
hive >   SELECT * FROM name_basics n
         WHERE n.birth_year = ( SELECT MAX(birth_year) FROM name_basics);
```

Solution

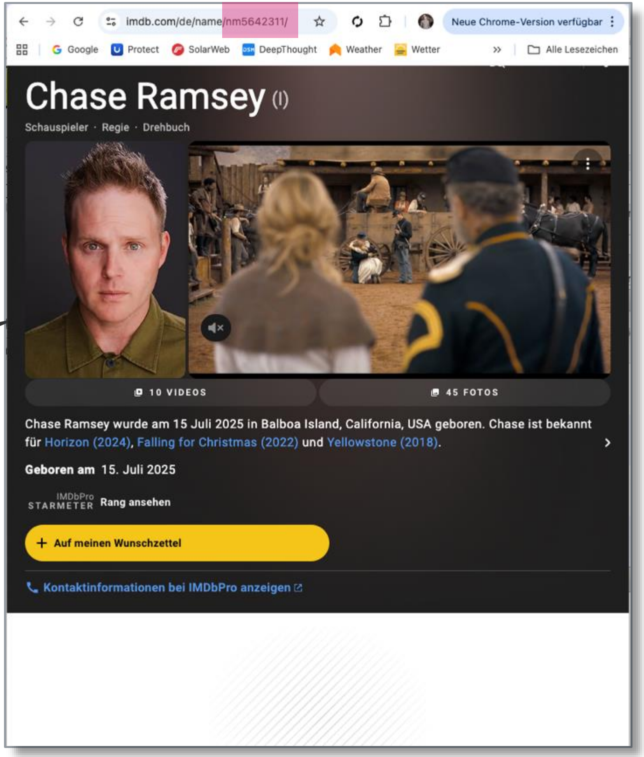
Exercise 5:

b) Who is the youngest actor/writer/… within the dataset?

```
hive >  SELECT * FROM name_basics n
        WHERE n.birth_year = ( SELECT MAX(birth_year)
                               FROM name_basics);
```

And it's Chase Ramsey, who is way older, so this one is actually shitty data in IMDB :D

```
nm5642311 Chase Ramsey 2025 NULL actor,director,writer tt17505010,tt14715170,tt4236770,tt17062324


Time taken: 65.166 seconds, Fetched: 1 row(s)
```



Solution

Exercise 5:

c) Create a list (`m.tconst, m.original_title, m.start_year, r.average_rating, r.num_votes`) of movies which are:
- equal or newer than year 2010
- have an average rating equal or better than 8,1
- have been voted more than 100.000 times

```
hive >   SELECT m.tconst, m.original_title, m.start_year, r.average_rating, r.num_votes
         FROM title_basics m JOIN title_ratings r on (m.tconst = r.tconst)
         WHERE r.average_rating >= 8.1 and m.start_year >= 2010 and m.title_type = 'movie'
         and r.num_votes > 100000
         ORDER BY r.average_rating desc, r.num_votes DESC;

tt1375666 Inception 2010 8.8 2735466
tt0816692 Interstellar 2014 8.7 2412377
tt23849204 12th Fail 2023 8.7 156943
tt15097216 Jai Bhim 2021 8.6 229573
tt10189514 Soorarai Pottru 2020 8.6 129634
tt1853728 Django Unchained 2012 8.5 1826339
tt6751668 Gisaengchung 2019 8.5 1103669
tt2582802 Whiplash 2014 8.5 1102686
tt1675434 Intouchables 2011 8.5 996020
tt15239678 Dune: Part Two 2024 8.5 669858
[…]
```

Solution

Exercise 5:

d) How many movies are in list of c)?

```
hive >    SELECT count(*)
          FROM title_basics m JOIN title_ratings r on (m.tconst = r.tconst)
          WHERE r.average_rating >= 8.1 and m.start_year >= 2010 and m.title_type = 'movie'
          and r.num_votes > 100000;

69
```

Solution

Exercise 5:

e) We want to know which years have been great for cinema.
Create a list with one row per year and a related count of movies which:
- have an average rating better than 8
- have been voted more than 100.000 times
ordered descending by count of movies.

```
hive >  SELECT m.start_year, count(*)
        FROM title_basics m JOIN title_ratings r on (m.tconst = r.tconst)
        WHERE r.average_rating > 8 AND m.title_type = 'movie'
        AND r.num_votes > 100000
        GROUP BY m.start_year
        ORDER BY count(*) DESC;

1995 8
2004 7
2018 6
2016 6
2015 6
[…]
```

Solution

Exercise 5:

So 1995 seems to be a really good year for cinema, 8 really good movies have been releases, but which are they?

```
hive > SELECT
            m.tconst, m.original_title, m.start_year, r.average_rating,
            r.num_votes
       FROM title_basics m JOIN title_ratings r ON (m.tconst = r.tconst)
       WHERE
            r.average_rating > 8 AND m.title_type = 'movie'
            AND r.num_votes > 100000 AND m.start_year = 1995
       ORDER BY r.average_rating DESC;

tt0114369 Se7en 1995 8.6 1954461
tt0114814 The Usual Suspects 1995 8.5 1208544
tt0113277 Heat 1995 8.3 775284
tt0114709 Toy Story 1995 8.3 1142717
tt0112573 Braveheart 1995 8.3 1141950
tt0112641 Casino 1995 8.2 603039
tt0112471 Before Sunrise 1995 8.1 366575
tt0113247 La haine 1995 8.1 217426

[…]
```

Introduction To The Challenges Of Distributed Data-Systems: Partitioning

Basics of Partitioning, Key-Range and Hash Partitioning, Partitioning of Secondary Indices, Rebalancing and Lookup of Partitions

Why Partitioning (and Replication)?

Partitioning is the process of continuously dividing data into subsets and distributing it to several nodes within a data-system. Usually each record or document within a partitioned data-system is distributed and directly assigned to certain partition. Partitioning serves the pur pose of, e.g.:

Scalability and Performance: Distributing data to multiple nodes, for instance increases read/write performance and throughput as read/write queries can be distributed to multiple nodes and handled concurrently. In this way it is possible parallelize IO (disk), computing power (CPU) as well as scale the memory usage needed to run a certain operation on a part of the dataset.

Low Latency: Using partitioning it is possible to place data close to where it is used (user or consumer applications).

Availability: Even if some nodes fail, only parts of the data are offline.

Replication vs Partitioning

	Replication	Partitioning
stores:	**copies** of the **same data** on **multiple nodes**	**subsets** (*partitions*) on **multiple nodes**
introduces:	**redundancy**	**distribution**
scalability:	**parallel IO**	**memory consumption**, certain parallel IO
availability:	**nodes** can **take** load of **failed nodes**	node **failures affect only parts** of the data

Different purposes, but usually used together

Partitioning – **Avoid Confusion On Terms**

To avoid confusion on the term partition or partitioning, let's list some other terms, you might have heard and which are frequently used synonymously:

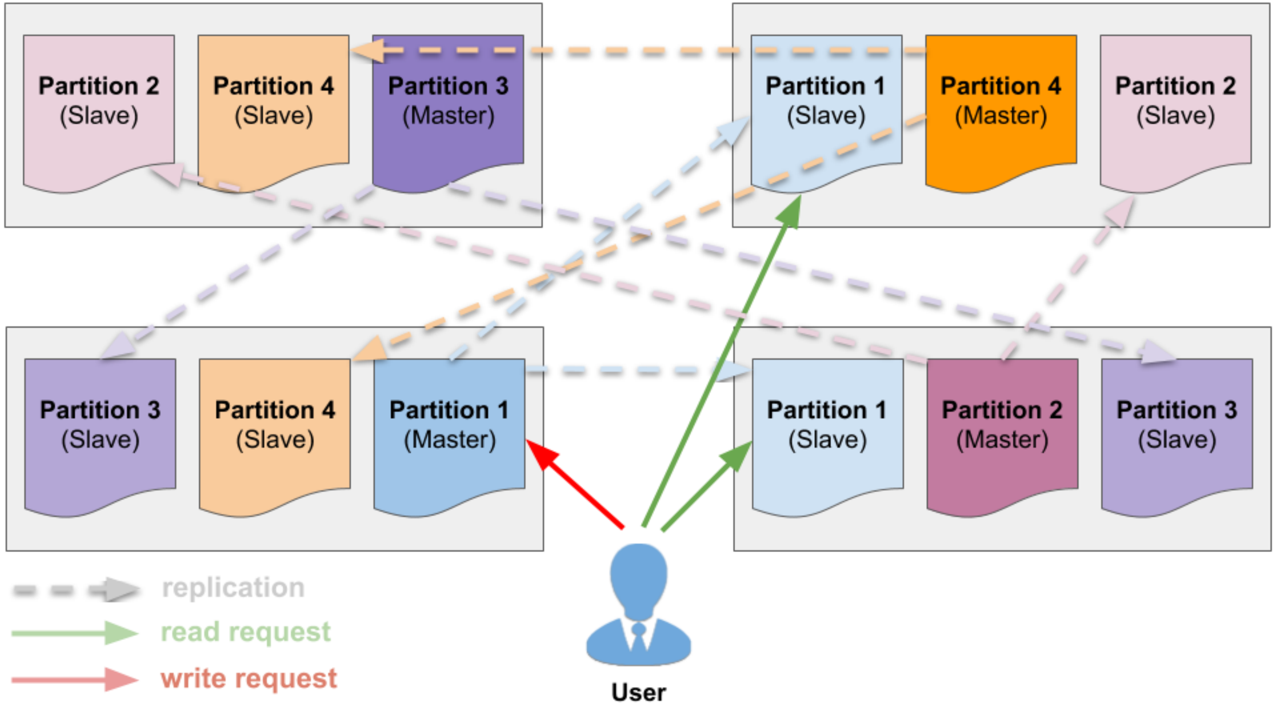
- ***shards/sharding*** - (e.g. *MongoDB*, *ElasticSearch* or *RethinkDB*)
- ***Vnodes/Virtual Nodes*** - (e.g. *Riak* or *Cassandra*)
- ***region*** - (e.g. *HBase*)
- ***tablet*** - (e.g. *BigTable*)
- ***vBucket*** - (e.g. *Couchbase*)

Partitioning and **Replication** are usually used together, especially when building data-intensive applications, as datasets are too big to be stored on a single server or replica, and benefits of replication are required (e.g. redundancy, fault-tolerance or high read/write through put). This can be achieved by storing partitions of a data set on multiple replica nodes.

Partitioning – **Avoid Confusion On Terms**

As **horizontal** *and* **vertical** *partitioning are mixed up sometimes, it is important to notice: when we speak about* ***partitioning*** *within this lecture, we mean* ***horizontal partioning****.* ***Vertical partitioning*** *is an* ***approach of traditional relational databases****, usually done by splitting datasets into multiple entities (e.g. tables or databases) and using references (e.g. to achieve normalization).*

Partitioning – **An Example**



- Partitioning and Replication
- Using Master-based Replication
- Each node is master for a certain partition
- Each partition has 2 slave nodes

→ **Ensure HA**

Partitioning – **Key-Value Data**

2 Purposes of Partitioning:

- **distributing a dataset**,
- more important: **distribute related load** (read/write queries) evenly among several nodes of a data-system

This requires:

- a **wise way of determining the partition** of a certain row or document → as it **directly affects the performance of a data-system**
- an **improper choosen distribution key** may cause:
  - **some nodes** to be **idle and/or empty** and
  - a **single node** to be the **processing bottleneck** and hitting its space limitations as all read/write requests end up on that single node
- an **appropriate distribution key** will **distribute the data evenly** and enable the data-system to (theoretically) scale linearily in terms of space utilization and request throug hput.

Partitioning – **Key-Value Data (Approaches)**

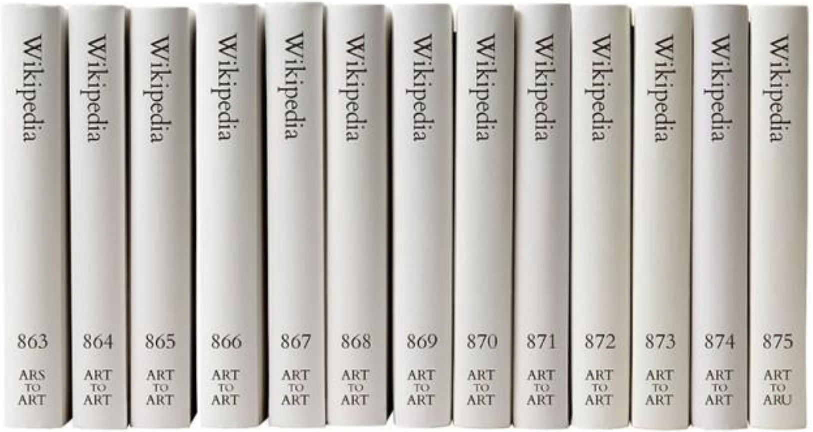
Key Range Partitioning: Derive a partition by determining whether a key is inside a certain value range.
→ *More details on next slides.*

Partitioning By Hash Value Of A Key: Derive a partition by a certain hash of a given key to achieve a mor e even data distribution. → *More details on next slides.*

Partitioning By List: Every partition to be used has an assigned list of values. A related partition is derived from the input dataset by checking whether it contains one of those values. For instance all rows containing iPhone, Samsung Galaxy and HTC One within a column `device_type` are assigned to partition Smart-phone. → *As no data-intensive system makes use of partitioning by list (as it is very improper to provide even data distributio n), we wont discuss this approach in detail.*

Round-Robin Partitioning: A very simple approach, which ensures even data distribution. For instance assignment to a partition can be achieved by **n** modulo **p** (*n = number of incoming data records, p = number of partitions*). → *As no data-intensive system makes use of round-robin partitioning (as for instance the direct access to an individual data record or subset usually requires accessing the whole dataset), we wont discuss this approach in detail.*

Partitioning – **Key-Range Partitioning**



Key Range partitioning is done by:

- **defining continues ranges of keys**
- **assigning each range to a certain partition**

→ If you are aware of the boundaries of each key range, you can easily derive a partition belonging to a certain data record (and in this way node of a data-system) just by using the key of the record.

→ This approach can be compared with an encyclopedia, which is partitioned into books, of which every-one stores a certain range of articles partitioned by the first letters of the name of the article.

→ For instance the article “*Arsenal F.C.*” will be found in partition 863 “*ARS*” - “*ART*”.

→ For instance used by: **RethinkDB** (called *ranged sharding*)

Partitioning – **Key-Range Partitioning**

Strengths:

- **Simple**
- **Range Lookups**

Weaknesses:

- **Datasets Are Changing:** A *key range* partitioning which was suitable in the past might not be appropriate in the future. Expensive rebalancing or even repartitioning might be needed somewhen.
  For instance web server log files partitioned per ranges of the URL `(/products/[A-B]`, `/products/[C-D]`... `/products/[Y-Z]`) maybe improper in the future, as some products will have heavier traffic than other products over time (***load skew***).

- **Hotspots:** Keys that seem very appropriate in terms of even distribution at first sight, e.g. partitioning of webserver logfiles over time (by using timestamp of data record), create hotspots as all write requests end up on the same partition (e.g. *today*), a single partition (and node(-s)) will underly heavy load whereas other partitions or rather nodes are idle (***load skew***).

- **Query Performance:** As you do not know the size of a partition beforehand, query performance is impredictable as well as partition pruning and partitionwise joins are more complex and less efficient.

Partitioning – **Hash Partitioning**

Hash partitioning is used to spread data efficiently and evenly among several certain partitions. This is achieved by:

→ splitting data in a randomized way
→ rather than by using information provided within the dataset (e.g. IDs) or
→ derived by arbitrary factors (e.g. time of data receival)

- The hash value itself is derived by a hash function (on a certain key of a data record) and is used to determine the partition a data records should be saved on.

Hash Function *is a function which takes input data of arbitrary size and usua lly provides an output of fixed size. The output of a hash function is called hash, hash value or digest. A hash function needs to be deterministic and uniform. Common use cases for hash functions are cryptography, checksums and partitioning.*

Partitioning – **Hash Partitioning** (Hash Functions)

Hash functions are commonly used for partitioning, as they are:

→ **deterministic** - as we need to be able to find records to be saved later on and
→ **uniform** - as we want to distribute the data as evenly as possible among the set of available partitions and nodes, even if the inputs of the hash function (e.g. data record key) are very similar.

Unlike cryptography, **partitioning makes use of hash functions** that are **not cryptographilly strong** (as this is not needed) but fast and less CPU consuming. Examples of commonly used hash function for distributed data-systems are:

- **MD5** - For instance used and supported by *MySQL* and *Cassandra.*
- **MurmurHash** - For instance supported by *Cassandra*.
- **SHA1** - For instance used and supported by *Riak*.
- **CRC32** - For instance used and supported by *Couchbase*.

Partitioning – **Hash Partitioning** (Hash Functions)

As an example for **determinism** and **uniformity** let’s take a quick look at **MD5** and how the hash value changes:

- when **a** single **character is added** to the input value,
- when just **a** single **character** of the input value is **changed** or
- the **same input** value is **hashed twice**

```
marcel$ md5 -s abc
MD5 ("abc") = 900150983cd24fb0d6963f7d28e17f72
// add a single character ("d")
marcel$ md5 -s abcd
MD5 ("abcd") = e2fc714c4727ee9395f324cd2e7f331f
// change a single character ("d" to "e")
marcel$ md5 -s abce
MD5 ("abce") = b9c4fe92c2a30ef69833ac8f53eebcec
// hash again with same input value
marcel$ md5 -s abce
MD5 ("abce") = b9c4fe92c2a30ef69833ac8f53eebcec
```

Code Snippet 2.7: Bash Output - *MD5 Hash For Several Input Values*

Partitioning – **Hash Partitioning (Hash Modulo)**

Hash Modulo Partitioning = take the calculated *hash value* **V** and calculate **`V modulo N`** (*number of partitions*).

→ This allows the the data-system to easily distribute and receive records to and from a given number of partitions.

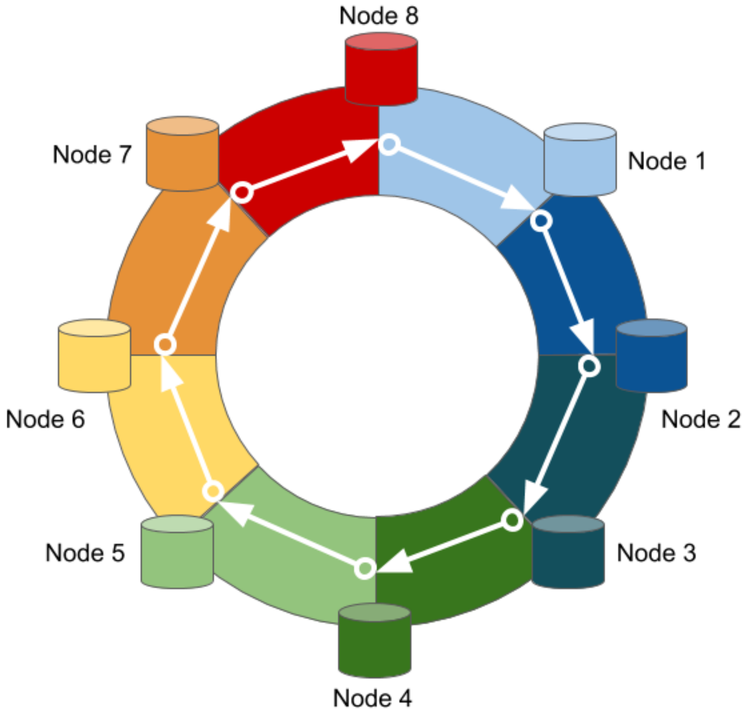
→ Major disadvantage in terms of **scalability** and **operability**.
→ Adding nodes results in different partition assignments for a lot of records (depending on size of N), which will require to **shuffle and reassign already saved data again** among all partitions.

- Nevertheless for instance **elasticsearch** makes use of it, a partition (called shard) is derived by: ***`shard = hash(routing) % number_of_primary_shards`***
- The number of shards for an *index* can increased (by ***`_split`***) or decreased (by ***`_shrink`***) some time after creation but this is not a trivial task and will usually require recreating the same or even creating a new index.

Partitioning – **Hash Partitioning (Consistent Hashing)**

- assign a *range of hashes* to every partition (and in this way node)
- every record will be stored and read from the partition in charge for a given *range of hash values*.
- imagine **consistent hashing** as a *ring of keys*
- each node (***Ni***) is in charge for serving all hash values ***v*** in between:
    - ***i*** and
    - the position ***j*** of its clockwise pre-decessor ***Nj***

`J < v < i`



Partitioning – **Hash Partitioning (Consistent Hashing)**

Strenghts:

- Adding/Removing a node → only *c/N* keys need to be re-distributed
  - *c* is the **count of hash values** and
  - *N* is the **number of nodes** within the data-system

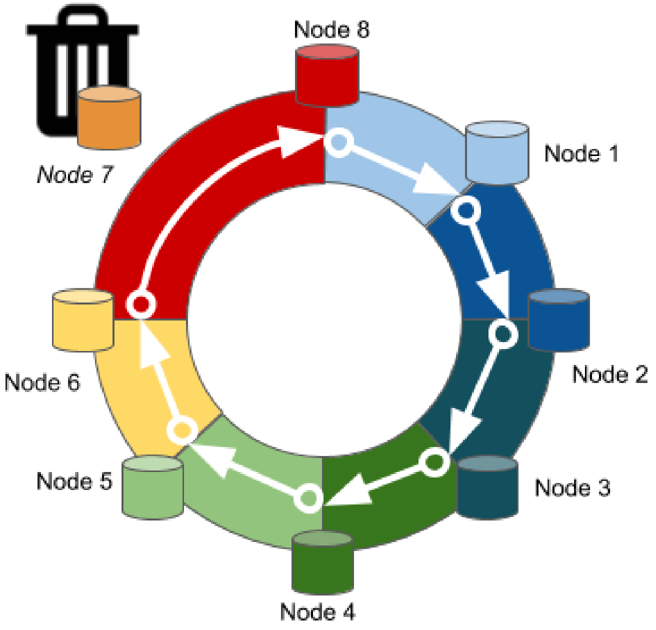
Weaknesses:

- Range Queries (as keys are randomly distributed amon data system)
- e.g. MongoDB provides *key-range partitioning* as well as *hash partitioning* to efficiently serve both use cases

Used by e.g.:

- Cassandra
- Riak
- VoldemortDB
- DynamoDB

Example: **Removing a Node**



Partitioning – **Partitioning Of Secondary Indices**

Secondary Index = Index in addition to primary index, which:

- is used to accelerates queries
- may not identify records uniquely
- used to lookup records with a certain value/attribute
- **needs to be partitioned as well**

→ Remember previous Facebook Profile example (Primary Key = User ID)
→ What if you want to acclerate the lookup of `cities` and `comanies`? Add a seco ndary index!

Partitioning – **Local Secondary Indices**

- **LSI** = *Local Secondary Index*
- every partition manages ist own index
- all pointers reference only to local data items

Strenths:

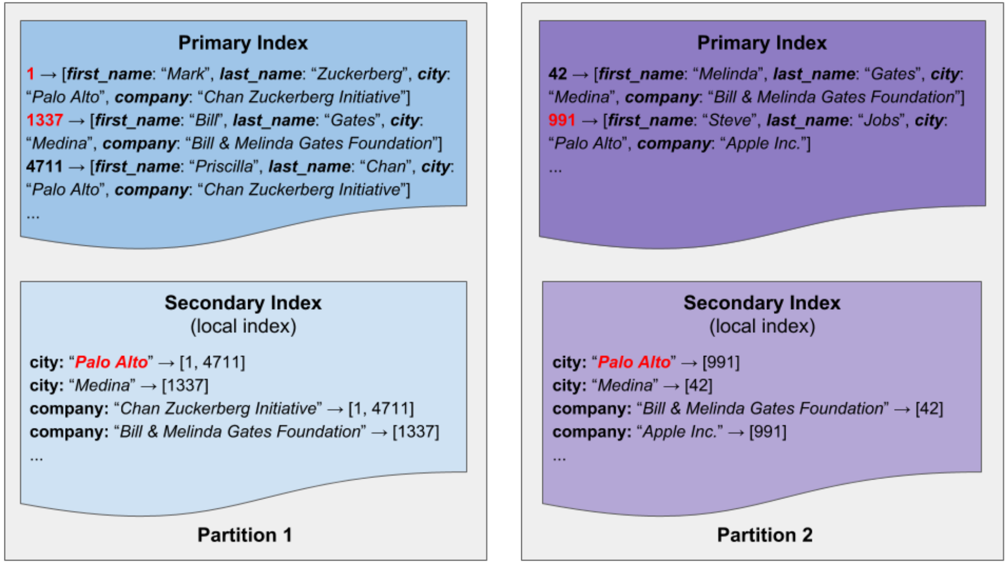
→ maintaining *local index* requires less over-head than *global index*

→ `INSERT`/`DELETE`/`UPDATE` performed locally

Weaknesses:

→ maintaining *local index* will compete with *local workload* and affect throughput

→ expensive `SELECT`: *scattering and gathering as well as related* overhead will probably affect read request performance



E.g. provided by:

- Riak
- Cassandra
- DynamoDB

Partitioning – **Global Secondary Indices**

- **GSI** = *Global Secondary Index*
- Index is partitioned, distributed and stored among several nodes independently of local data records
- pointers reference to local but also remote data records

Strenths:

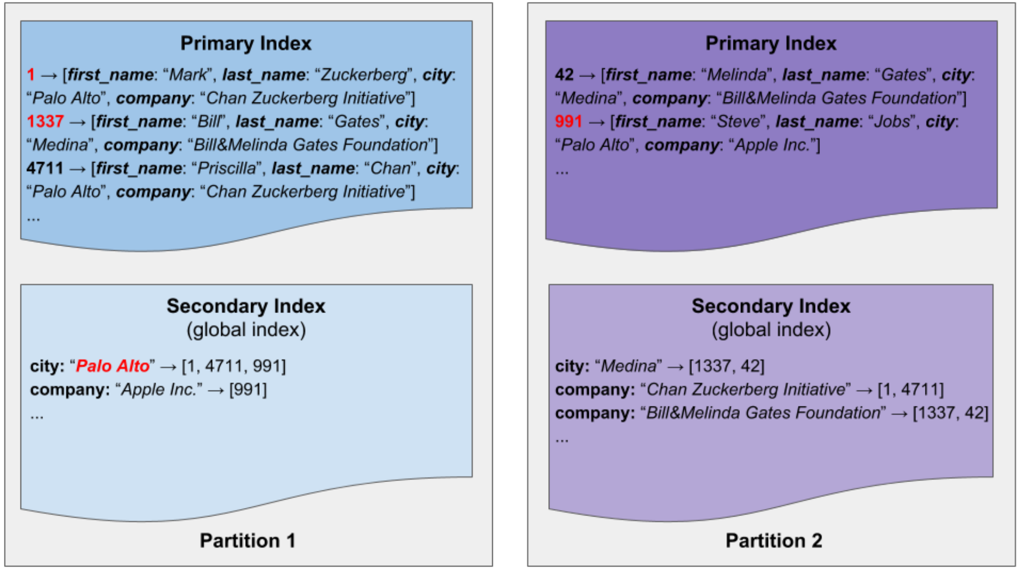
→ `SELECT` statements need to query only one index partition

→ No *scattering and gathering (like using LSI)*

Weaknesses:

→ maintaining *global index* requires more overhead than a *local index* → *`INSERT/DELETE/UPDATE`* statements require remote updates

→ usually weakens read-consistency as indices updates take more time → DynamoDB ***eventual consistency***



E.g. provided by:

- Oracle
- MS SQL Server
- Couchbase
- DynamoDB

Partitioning – **Rebalancing Partitions**

Why?

- Node Failure → other nodes need to take over
- Query load increases → more CPUs and RAM required
- Data Size increases → more RAM and disk space required

Goals:

- Minimal Data Shuffle: Move as less data as possible around nodes during rebalancing
- Evenly Distribution: Data distribution should be even after rebalacning.
- No Availability Impact: Rebalancing should have as less impact as possible on a running data-system, requiring no downtime.

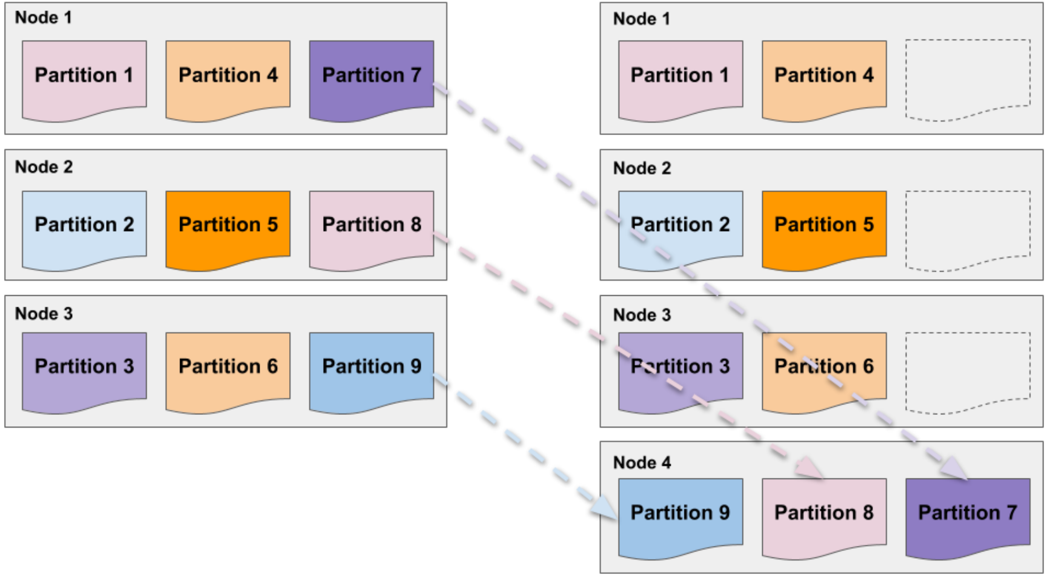
Partitioning – **Rebalancing Partitions (Hash Modulo)**

Approach:
- hash % n

Contra:
- stupid
- requires shuffling a lot of data, as assignment of partitions to nodes changes significantly

Used by e.g.:
- shouldn't be used

Partitioning – **Rebalancing Partitions (Fixed Number of Partitions)**



Approach:

- Create more partitions than there are nodes on initial setup
- new nodes „steal" partitions from old ones

Pro/Contra:

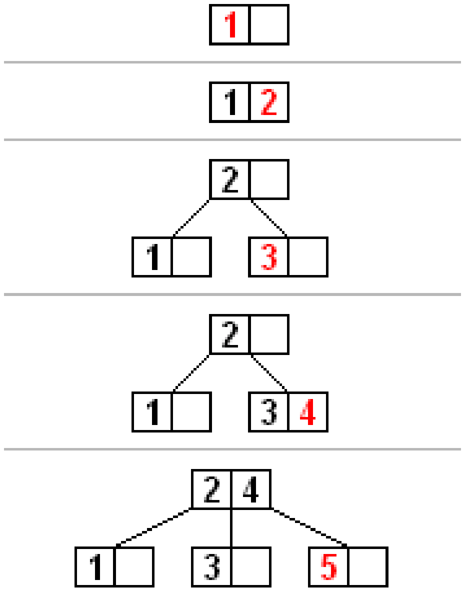
- partition/node mappings change
- but for less partitions

→ only partial rewrite of partitions

Used by e.g.:

- Voldemort
- Riak
- ElasticSearch
- Couchbase

Partitioning – Rebalancing Partitions (Dynamic No. Of Partitions)



Approach:

- Create certain number of initial partitions
- Idea similar to B-Trees

→ If a partition exceeds a threshold in size, it gets split

→ If a partition falls below a threshold in size, merge it with another

- new nodes „steal“ partitions from old ones

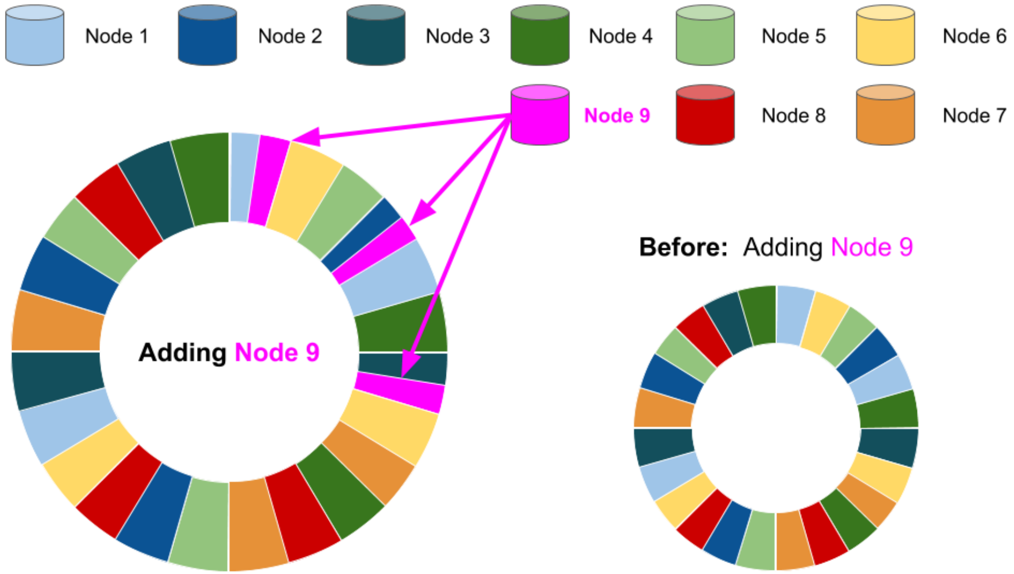
Pro/Contra:

- evenly distribution
- continues overhead during runtime

Used by e.g.:

- MongoDB
- RethinkDB

Partitioning – **Rebalancing Partitions (Fixed Partition No. Per Node)**



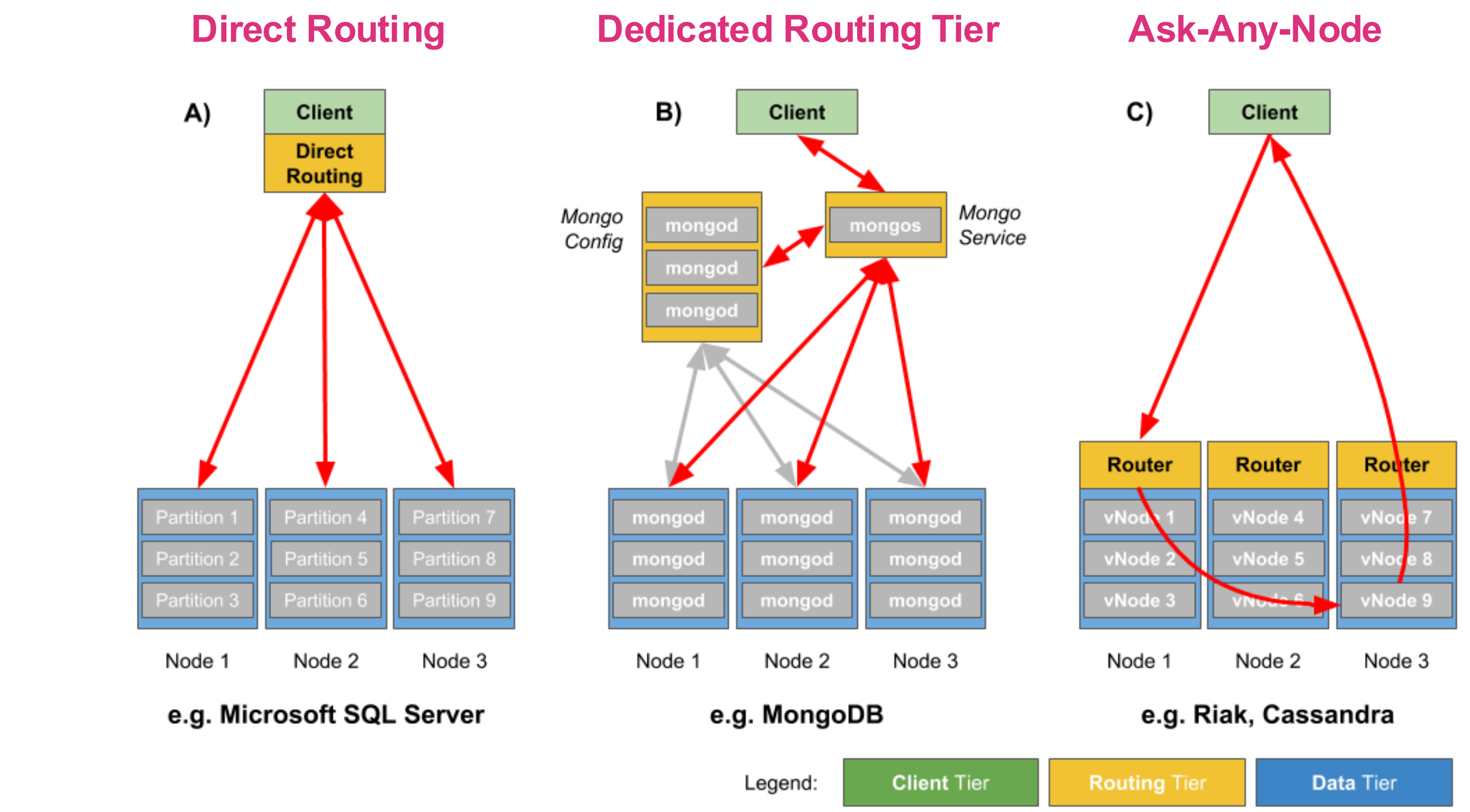
Approach:
- create fixed number of partitions at initial setup
- new nodes randomly split partitions of old nodes

→ steal half of partitions of old nodes

Pro/Contra:
- works well for range partitioning (hash or keys)
- load may be temprorarily unbalanced but will be even again over time

Used by e.g.:
- Cassandra

Partitioning – **Partition Lookup**



www.marcel-mittelstaedt.com

Break

TIME FOR
A
BREAK

Exercises Preparation

Setup HiveServer2 For Remote Connections

Start Gcloud VM and Connect

1. Start Gcloud Instance:

```
gcloud compute instances start big-data
```

2. Connect to Gcloud instance via SSH (on Windows using Putty):

```
ssh hans.wurst@XXX.XXX.XXX.XXX
```

Pull and Start Docker Container

1. Pull Docker Image:

```
docker pull marcelmittelstaedt/hiveserver_base:latest
```

2. Start Docker Image:

```
docker run -dit --name hiveserver_base_container \
    -p 8088:8088 -p 9870:9870 -p 9864:9864 \
    -p 10000:10000 -p 9000:9000 \
    marcelmittelstaedt/hiveserver_base:latest
```

3. Wait till first Container Initialization finished:

```
docker logs hiveserver_base_container

[…]
Stopping nodemanagers
Stopping resourcemanager
Container Startup finished.
```

Start Hadoop Cluster

1. Get into Docker container:

```
docker exec -it hiveserver_base_container bash
```

2. Switch to hadoop user:

```
sudo su hadoop
```

```
cd
```

3. Start Hadoop Cluster:

```
start-all.sh
```

Start HiveServer2

1. Start HiveServer2 (takes some time!), wait till you see:

```
hive/bin/hiveserver2

2021-02-21 16:43:55: Starting HiveServer2
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hadoop/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4
j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hadoop/hadoop/share/hadoop/common/lib/slf4j-log4j12-1
.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive Session ID = ae41ac72-4dbd-4115-9863-59c3859c3db6
Hive Session ID = 17f9f63b-4018-4976-bb7d-15fbf1bc8042
Hive Session ID = 83b2ad76-c248-46a1-91d4-f2ad289614ee
Hive Session ID = b9ff1fd3-ccb1-4254-abc7-4c696d8ff8a1
[…]
```

Connect To HiveServer2 via JDBC

1. Download JDBC SQL Client, e.g. *DBeaver*:

Mac OSX:

```
wget https://dbeaver.io/files/dbeaver-ce-latest-macos.dmg
```

Linux (Debian):

```
wget https://dbeaver.io/files/dbeaver-ce_latest_amd64.deb
```

Linux (RPM):

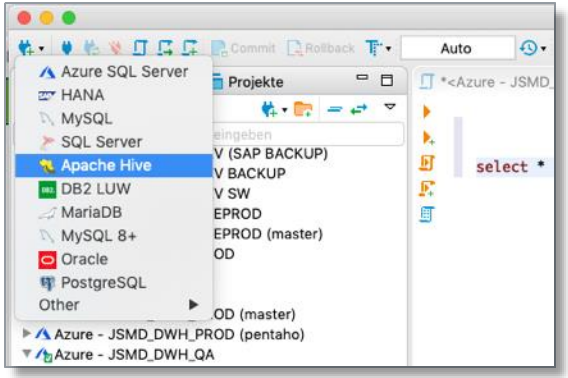
```
wget https://dbeaver.io/files/dbeaver-ce-latest-stable.x86_64.rpm
```

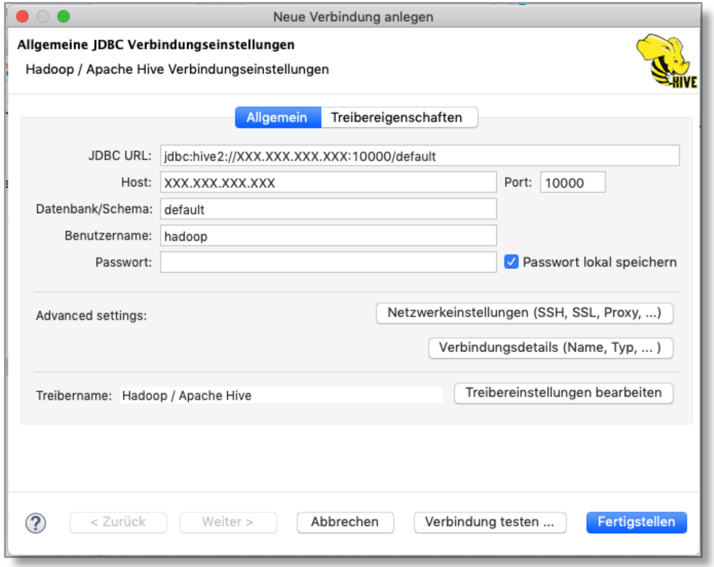
Windows:

```
wget https://dbeaver.io/files/dbeaver-ce-latest-x86_64-setup.exe
```

Connect To HiveServer2 via JDBC

2. Configure Connection To Hive Server:





Let's get some data…

1. Get some IMDb data:

```
wget https://datasets.imdbws.com/title.basics.tsv.gz && gunzip title.basics.tsv.gz
wget https://datasets.imdbws.com/title.ratings.tsv.gz && gunzip title.ratings.tsv.gz
wget https://datasets.imdbws.com/name.basics.tsv.gz && gunzip name.basics.tsv.gz
```

2. Put it into HDFS:

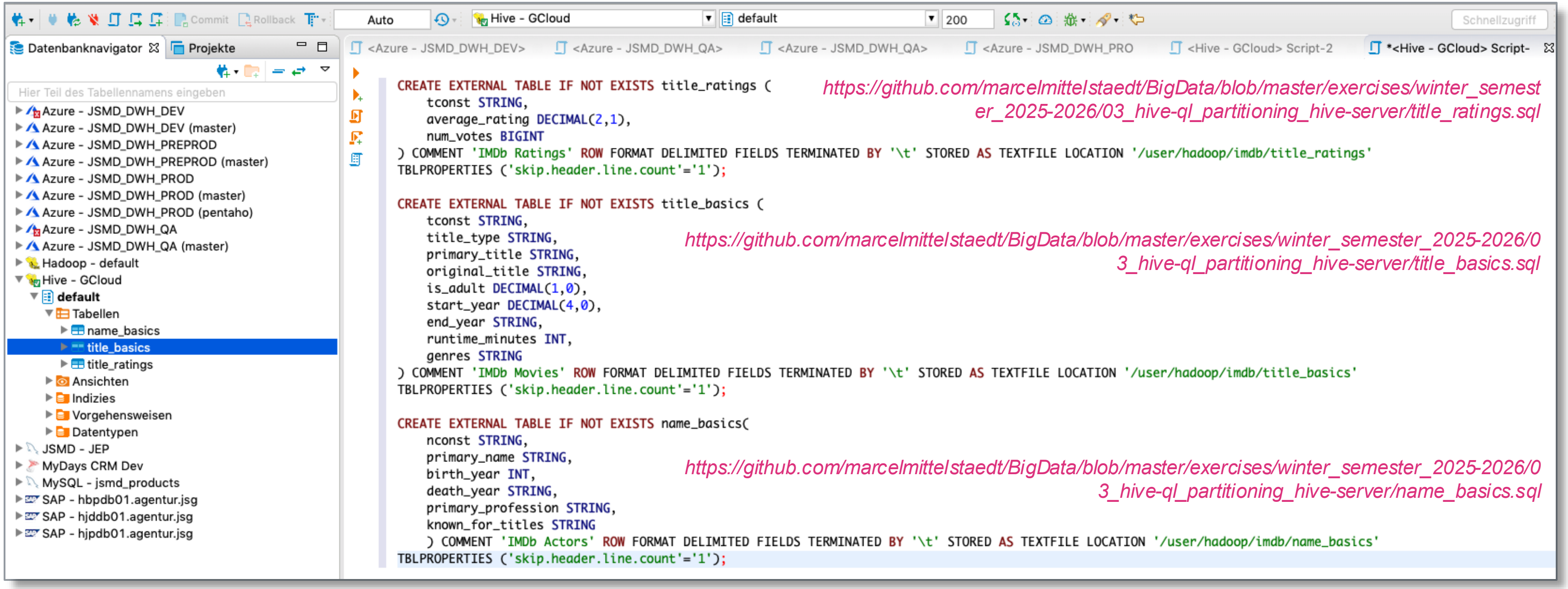
```
hadoop fs -mkdir /user/hadoop/imdb
```

```
hadoop fs -mkdir /user/hadoop/imdb/title_basics && hadoop fs -mkdir /user/hadoop/imdb/title_r
atings && hadoop fs -mkdir /user/hadoop/imdb/name_basics
```

```
hadoop fs -put title.basics.tsv /user/hadoop/imdb/title_basics/title.basics.tsv && hadoop fs
-put title.ratings.tsv /user/hadoop/imdb/title_ratings/title.ratings.tsv && hadoop fs -put na
me.basics.tsv /user/hadoop/imdb/name_basics/name.basics.tsv
```

Create some external tables…

1. Create some tables on top of files:



Query some data…

1. Query some data:

```sql
SELECT
    m.tconst,
    m.original_title,
    m.start_year,
    r.average_rating,
    r.num_votes
FROM
    title_basics m
    JOIN title_ratings r ON (m.tconst = r.tconst)
WHERE
    r.average_rating >= 8.1 AND m.start_year >= 2010
    AND m.title_type  = 'movie' AND r.num_votes > 100000
ORDER BY r.average_rating DESC, r.num_votes DESC;
```

Ergebnisse 1

SELECT m.tconst, m.original_title, m.start_year, | Geben Sie einen SQL-Ausdruck ein, um die Ergebnisse zu filtern (verwenden Si

	tconst	original_title	start_year	average_rating	num_votes
1	tt1375666	Inception	2.010	8,8	2.738.181
2	tt0816692	Interstellar	2.014	8,7	2.415.719
3	tt23849204	12th Fail	2.023	8,7	157.437
4	tt15097216	Jai Bhim	2.021	8,6	229.927
5	tt10189514	Soorarai Pottru	2.020	8,6	129.952
6	tt1853728	Django Unchained	2.012	8,5	1.828.304
7	tt6751668	Gisaengchung	2.019	8,5	1.105.714
8	tt2582802	Whiplash	2.014	8,5	1.104.248
9	tt1675434	Intouchables	2.011	8,5	997.037
10	tt15239678	Dune: Part Two	2.024	8,5	671.701

Break

TIME FOR

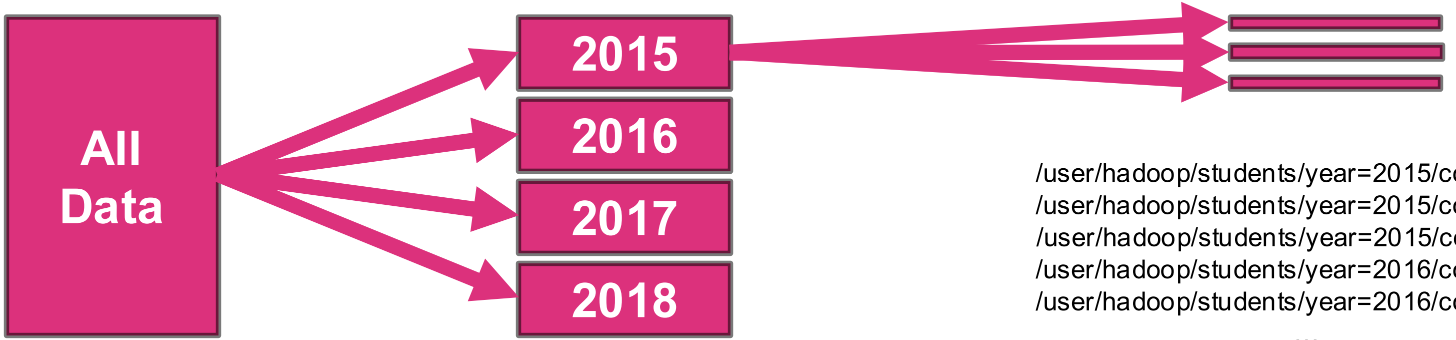
A

BREAK

HandsOn –
Data Partitioning with HDFS and Hive (via JDBC)

HDFS/Hive - **Partitioning**

- Partitioning of data distributes load and speeds up data processing
- A table can have one or more partition columns, defined by the time of creating a table (`CREATE TABLE student(id Int, name STRING) PARTITIONED BY ( year STRING)… STORED AS TEXTFILE LOCATION '/user/hadoop/students'`)
- partitioning can be done either **static** or **dynamic**
- each distinct value of a partition column is represented by a **HDFS directory**



/user/hadoop/students/

/user/hadoop/students/year=2015/
/user/hadoop/students/year=2016/
…

/user/hadoop/students/year=2015/country=de/
/user/hadoop/students/year=2015/country=en/
/user/hadoop/students/year=2015/country=us/
/user/hadoop/students/year=2016/country=de/
/user/hadoop/students/year=2016/country=en/
…

Static Partitioning – **Create Partitioned Table**

1. Create partitioned version of table `imdb_ratings`: **`imdb_ratings_partitioned`:**

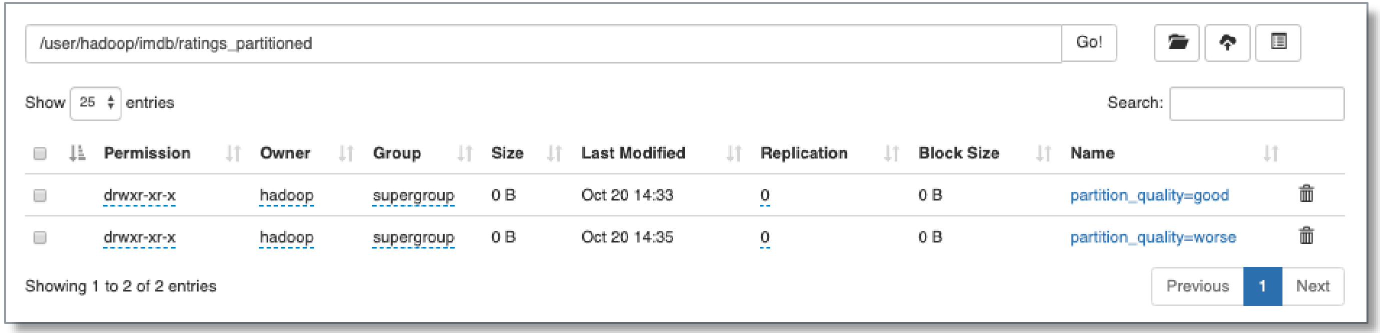
```
CREATE TABLE IF NOT EXISTS title_ratings_partitioned(
    tconst STRING,
    average_rating DECIMAL(2,1),
    num_votes BIGINT
) PARTITIONED BY (partition_quality STRING)
STORED AS PARQUET LOCATION '/user/hadoop/imdb/ratings_partitioned';
```

Static Partitioning – **INSERT Into Table via Hive**

1. Migrate and partition data of table `title_ratings` to table **`title_ratings_partitioned`**:

```
INSERT OVERWRITE TABLE title_ratings_partitioned PARTITION(partition_quality='good')
SELECT r.tconst, r.average_rating, r.num_votes FROM title_ratings r WHERE r.average_rating >= 7;

INSERT OVERWRITE TABLE title_ratings_partitioned PARTITION(partition_quality='worse')
SELECT r.tconst, r.average_rating, r.num_votes FROM title_ratings r WHERE r.average_rating < 7;
```

2. Check Success on HDFS:



/user/hadoop/imdb/ratings_partitioned Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	hadoop	supergroup	0 B	Oct 20 14:33	0	0 B	partition_quality=good
drwxr-xr-x	hadoop	supergroup	0 B	Oct 20 14:35	0	0 B	partition_quality=worse

Showing 1 to 2 of 2 entries

Previous 1 Next

Static Partitioning – **INSERT Into Table via Hive**

3. Check Success via Hive:

```sql
select distinct average_rating from title_ratings_partitioned where partition_quality = 'good';
```

Result

select distinct average_rating from title_ratings_par | Geben Sie einen SQL-Ausdruck ein, um die Ergebnisse zu filtern (verwend

	average_rating
1	7,0
2	7,1
3	7,2
4	7,3
5	7,4
6	7,5
7	7,6
8	7,7
9	7,8

Dynamic Partitioning – **Create Partitioned Table**

1. Create partitioned version of table `title_basics`: **`title_basics_partitioned`**:

```
CREATE TABLE IF NOT EXISTS title_basics_partitioned(
    tconst STRING,
    title_type STRING,
    primary_title STRING,
    original_title STRING,
    is_adult DECIMAL(1,0),
    start_year DECIMAL(4,0),
    end_year STRING,
    runtime_minutes INT,
    genres STRING
) PARTITIONED BY (partition_year DECIMAL(4,0)) STORED AS PARQUET L
OCATION '/user/hadoop/imdb/title_basics_partitioned';
```

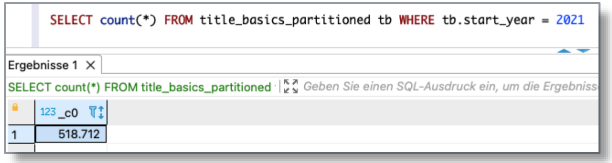
Dynamic Partitioning – **INSERT Into Table via Hive**

1. Migrate and partition data of table `title_basics` to table **`title_basics_partitioned`**:

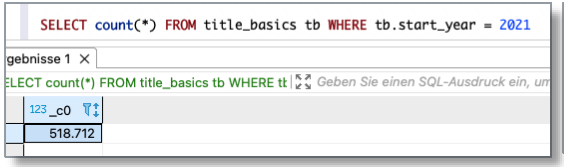
```
set hive.exec.dynamic.partition.mode=nonstrict; -- enable dynamic partitioning

INSERT OVERWRITE TABLE title_basics_partitioned partition(partition_year)
SELECT t.tconst, t.title_type, t.primary_title, t.original_title, t.is_adult,
t.start_year, t.end_year, t.runtime_minutes, t.genres,
t.start_year -- last column = partition column
FROM title_basics t;
```

2. Check Success via Hive:

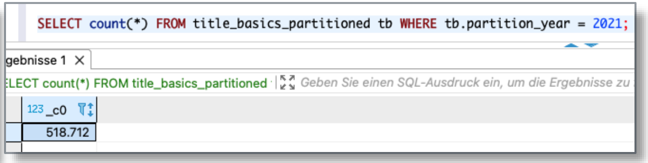
Runtime: **28 seconds**



Runtime: **28 seconds**



Runtime: **<1 second**



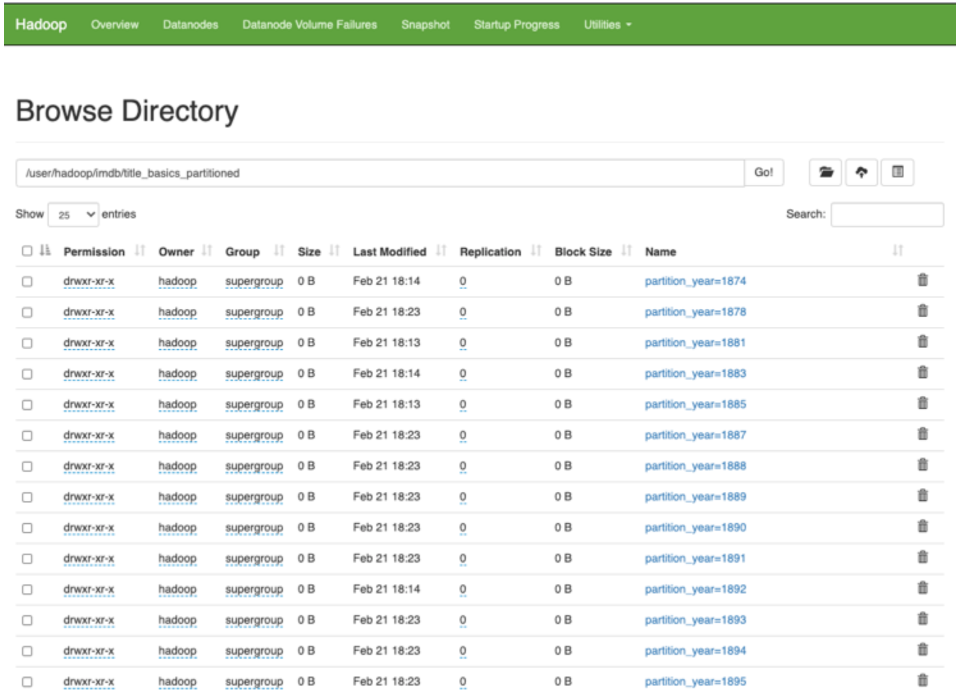
Dynamic Partitioning – **INSERT Into Table via Hive**

3. Check Success on HDFS:

```
hadoop fs -ls /user/hadoop/imdb/title_basics_partitioned

Found 149 items
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:14 /user/hadoop/imdb/title_basics_partitioned/partition_year=1874
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:23 /user/hadoop/imdb/title_basics_partitioned/partition_year=1878
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:13 /user/hadoop/imdb/title_basics_partitioned/partition_year=1881
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:14 /user/hadoop/imdb/title_basics_partitioned/partition_year=1883
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:13 /user/hadoop/imdb/title_basics_partitioned/partition_year=1885
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:23 /user/hadoop/imdb/title_basics_partitioned/partition_year=1887
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:23 /user/hadoop/imdb/title_basics_partitioned/partition_year=1888
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:23 /user/hadoop/imdb/title_basics_partitioned/partition_year=1889
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:23 /user/hadoop/imdb/title_basics_partitioned/partition_year=1890
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:23 /user/hadoop/imdb/title_basics_partitioned/partition_year=1891
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:14 /user/hadoop/imdb/title_basics_partitioned/partition_year=1892
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:23 /user/hadoop/imdb/title_basics_partitioned/partition_year=1893
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:23 /user/hadoop/imdb/title_basics_partitioned/partition_year=1894
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:23 /user/hadoop/imdb/title_basics_partitioned/partition_year=1895
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:23 /user/hadoop/imdb/title_basics_partitioned/partition_year=1896
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:23 /user/hadoop/imdb/title_basics_partitioned/partition_year=1897
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:23 /user/hadoop/imdb/title_basics_partitioned/partition_year=1898
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:22 /user/hadoop/imdb/title_basics_partitioned/partition_year=1899
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:22 /user/hadoop/imdb/title_basics_partitioned/partition_year=1900
drwxr-xr-x   - hadoop supergroup          0 2021-02-21 17:22 /user/hadoop/imdb/title_basics_partitioned/partition_year=1901
[…]
```

Dynamic Partitioning – **INSERT Into Table via Hive**

4. Check Success via HDFS Web Browser (http://X.X.X.X:9870/)


Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Browse Directory

/user/hadoop/imdb/title_basics_partitioned Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	hadoop	supergroup	0 B	Feb 21 18:14	0	0 B	partition_year=1874
drwxr-xr-x	hadoop	supergroup	0 B	Feb 21 18:23	0	0 B	partition_year=1878
drwxr-xr-x	hadoop	supergroup	0 B	Feb 21 18:13	0	0 B	partition_year=1881
drwxr-xr-x	hadoop	supergroup	0 B	Feb 21 18:14	0	0 B	partition_year=1883
drwxr-xr-x	hadoop	supergroup	0 B	Feb 21 18:13	0	0 B	partition_year=1885
drwxr-xr-x	hadoop	supergroup	0 B	Feb 21 18:23	0	0 B	partition_year=1887
drwxr-xr-x	hadoop	supergroup	0 B	Feb 21 18:23	0	0 B	partition_year=1888
drwxr-xr-x	hadoop	supergroup	0 B	Feb 21 18:23	0	0 B	partition_year=1889
drwxr-xr-x	hadoop	supergroup	0 B	Feb 21 18:23	0	0 B	partition_year=1890
drwxr-xr-x	hadoop	supergroup	0 B	Feb 21 18:23	0	0 B	partition_year=1891
drwxr-xr-x	hadoop	supergroup	0 B	Feb 21 18:14	0	0 B	partition_year=1892
drwxr-xr-x	hadoop	supergroup	0 B	Feb 21 18:23	0	0 B	partition_year=1893
drwxr-xr-x	hadoop	supergroup	0 B	Feb 21 18:23	0	0 B	partition_year=1894
drwxr-xr-x	hadoop	supergroup	0 B	Feb 21 18:23	0	0 B	partition_year=1895

Break

TIME FOR
A
BREAK

Exercise

HDFS/Hive: Work with Partitions

HDFS/Hive Partitioning Exercises - **IMDB**

1. Execute Tasks of previous HandsOn Slides

2. Create a (*statically*) partitioned table **`name_basics_partitioned`**, which:
   - contains all columns of table `name_basics`
   - is statically partitioned by `partition_is_alive`, containing:
     - „*alive*" in case actor is still alive
     - „*dead*" in case actor is already dead

   Load all data from `name_basics` into table **`name_basics_partitioned`**

3. Create a (*dynamically*) partitioned table **`imdb_movies_and_ratings_partitioned`**, which:
   - contains all columns of the two tables `title_basics` and `title_ratings` and
   - is partitioned by start year of movie (create and add column `partition_year`).

   Load all data of `title_basics` and `title_ratings` into table:
   `imdb_movies_and_ratings_partitioned`

Well Done

WE'RE DONE FOR ...TODAY

Stop Your VM Instances


```
gcloud compute instances stop big-data
```