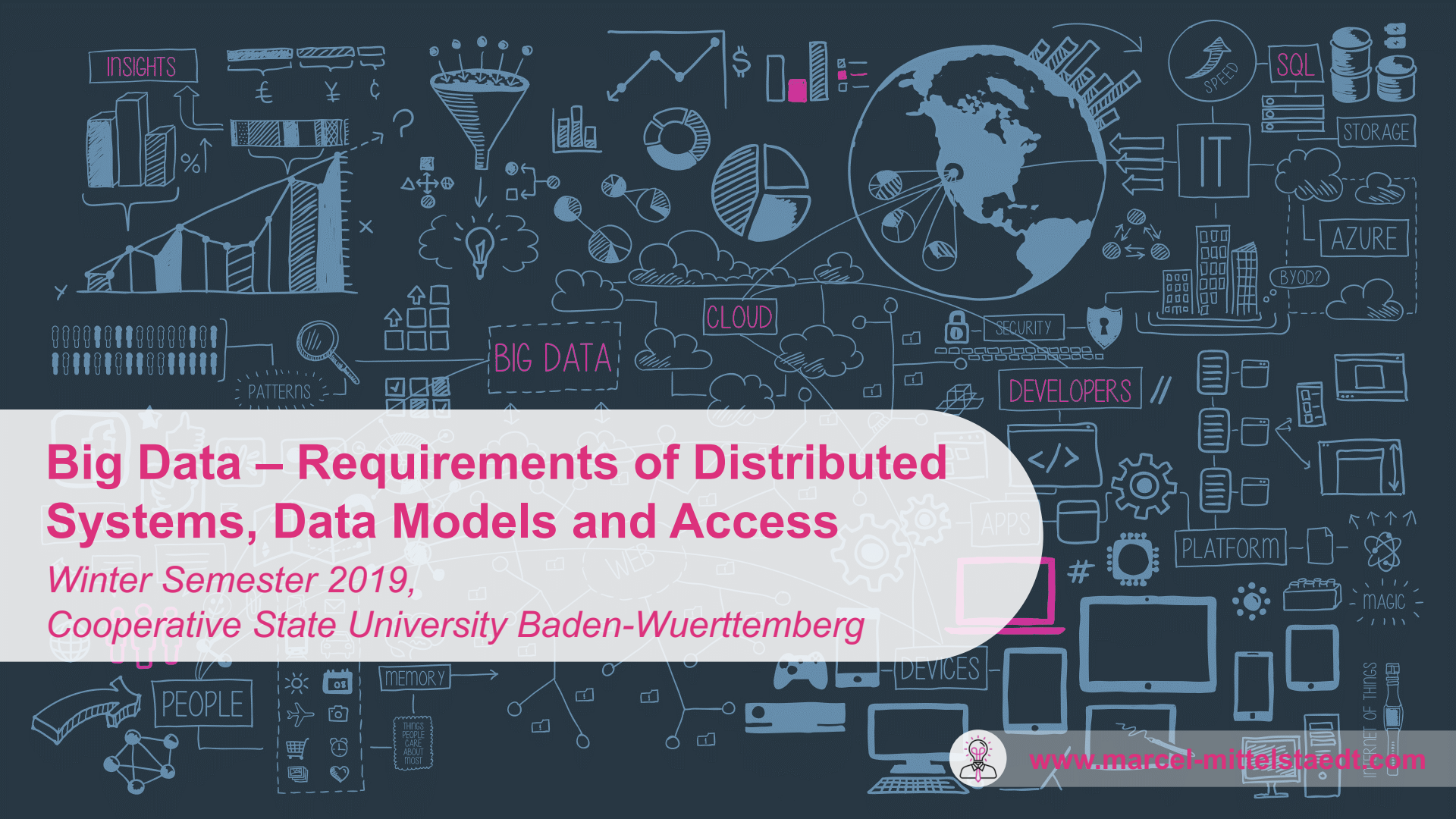


Big Data – Requirements of Distributed Systems, Data Models and Access

Winter Semester 2019,
Cooperative State University Baden-Wuerttemberg

www.marcel-mittelstaedt.com

Agenda – 21.10.2019

01

Presentation and Discussion: Exercises Of Last Lecture

Hadoop, HDFS, YARN, MapReduce Example, Hive, HiveQL

02

Introduction To Distributed Systems, Data Models and Access

(Non-)Functional Requirements Of Distributed Data-Systems, Data Models And Access

03

HandsOn – Java MapReduce, Hive, Partitions and Hive via JDBC

Quick Introduction To The MapReduce Programming Paradigm, Java MapReduce, Hive, HiveQL, HDFS/Hive Partitions and Hive via JDBC (HiveServer2)

04

Exercise

Write, Compile and Run MapReduce Job in Java, HDFS/Hive Partitioning, HiveServer2 and Hive/HiveQL via JDBC

Schedule

	Lecture Topic	*HandsOn*
14.10.2019 13:15-18:00 Ro. 1.18	About This Lecture, Introduction to Big Data, Setup Cloud Environment (Google Cloud)	HandsOn Hadoop, Hive and Hive(-QL)
21.10.2019 13:15-18:00 Ro. 1.18	**(Non-)Functional Requirements Of Distributed Data-Systems, Data Models and Access**	Partitioning with HDFS/Hive, HiveServer2
28.10.2019 13:15-18:00 Ro. 1.18	Challenges Of Distributed Data Systems: Replication and Partitioning	Spark, Scala and PySpark/Jupyter
04.11.2019 13:15-18:00 Ro. 1.18	ETL Workflow and Automation, Batch Processing	Pentaho Data Integration & Airflow
11.11.2019 13:15-18:00 Ro. 1.18	Practical Exam	Work On Practical Exam
18.11.2019 13:15-18:00 Ro. 1.18	Practical Exam Presentation	

Solution – Exercise I

Hadoop, HDFS, YARN, MapReduce Example

Solution

Prerequisites:

- install Ubuntu 18.04
- Install Java JDK 1.8.0
- Create Hadoop user
- Install and Setup SSH (*public/private key authentication, authorized_keys, …*)
- Install and Configure Hadoop 3.1.3 (*pseudo-distributed mode*)
- Start HDFS and YARN
- Clone Git Repo:

```
git clone https://github.com/marcelmittelstaedt/BigData.git
```


Solution

Exercise 2:

1. Copy sample file from GIT repo to HDFS user directory:

```
hadoop fs -put BigData/exercises/winter_semester_2019-2020/01_hadoop/sample_data/Faust_1.txt /user/hadoop/Faust_1.txt
```

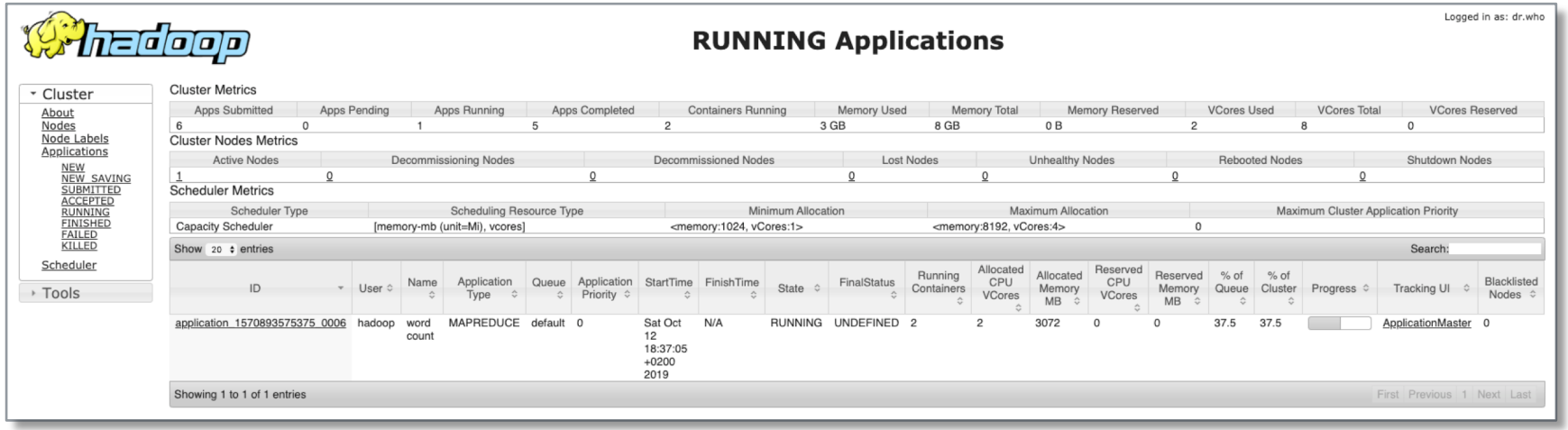
2. Use and run default MapReduce Jar (*hadoop-mapreduce-examples-3.1.2.jar*) to calculate **wordcount** for text file „*Faust_1.txt*“.

```
hadoop jar hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.2.jar wordcount /user/hadoop/Faust_1.txt /user/hadoop/Faust_1_Output

[…]
2019-10-12 16:37:06,033 INFO mapreduce.Job: Running job: job_1570893575375_0006
2019-10-12 16:37:12,157 INFO mapreduce.Job: Job job_1570893575375_0006 running in uber mode : false
2019-10-12 16:37:12,158 INFO mapreduce.Job:  map 0% reduce 0%
2019-10-12 16:37:17,236 INFO mapreduce.Job:  map 100% reduce 0%
2019-10-12 16:37:22,279 INFO mapreduce.Job:  map 100% reduce 100%
2019-10-12 16:37:23,295 INFO mapreduce.Job: Job job_1570893575375_0006 completed successfully
[…]
```

Solution

Exercise 2:

3. Take a look at Ressource Manager for Job Execution (http://XXX.XXX.XXX.XXX:8088/cluster/apps/RUNNING):



Solution

Exercise 2:

4. a) Copy MapReduce output file back to ubuntu local filesystem (using bash):

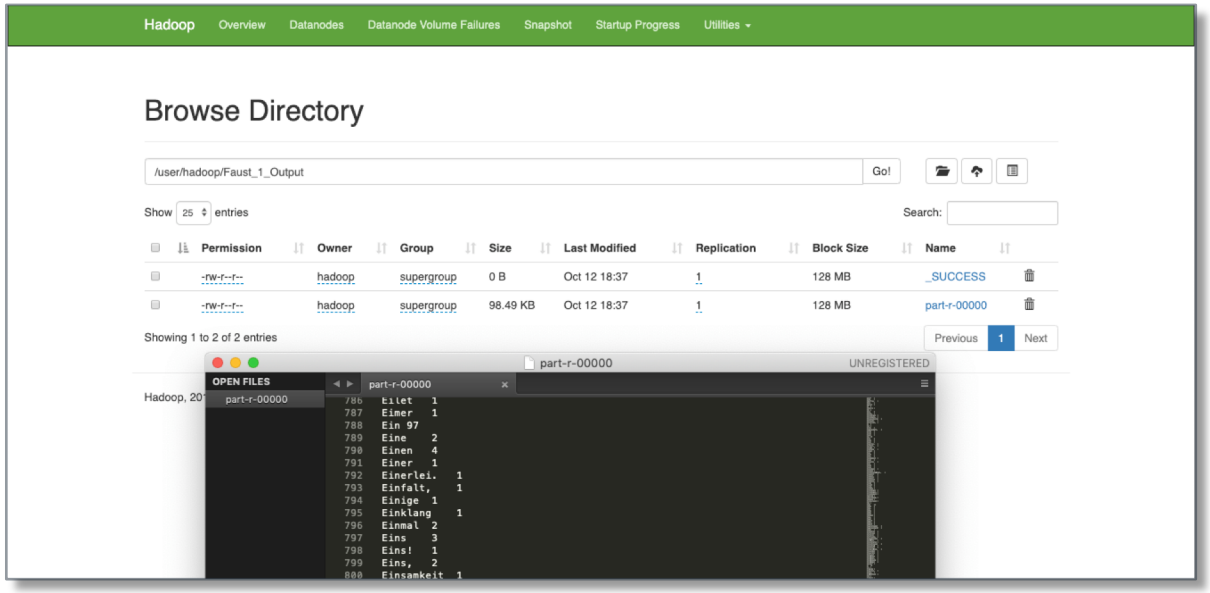
```
hadoop fs -get /user/hadoop/Faust_1_Output/part-r-00000 Faust_1_Output.csv
```

```
shuf -n 10 Faust_1_Output.csv

Phantasie,          1
unwanden. 1
winden,   1
Offenbarung,        2
Undene!   1
Winternächte        1
derweil   1
wiederholten        1
tun       3
Gestalten.          1
```

Solution

Exercise 2:

4. b) Copy MapReduce output file back to ubuntu local filesystem (using Web Filebrowser):



Solution

Exercise 3:

1. Copy sample file from GIT repo to HDFS user directory:

```
hadoop fs -put BigData/exercises/winter_semester_2019-2020/01_hadoop/sample_data/Faust_1.txt /user/hadoop/Faust_1.txt
```

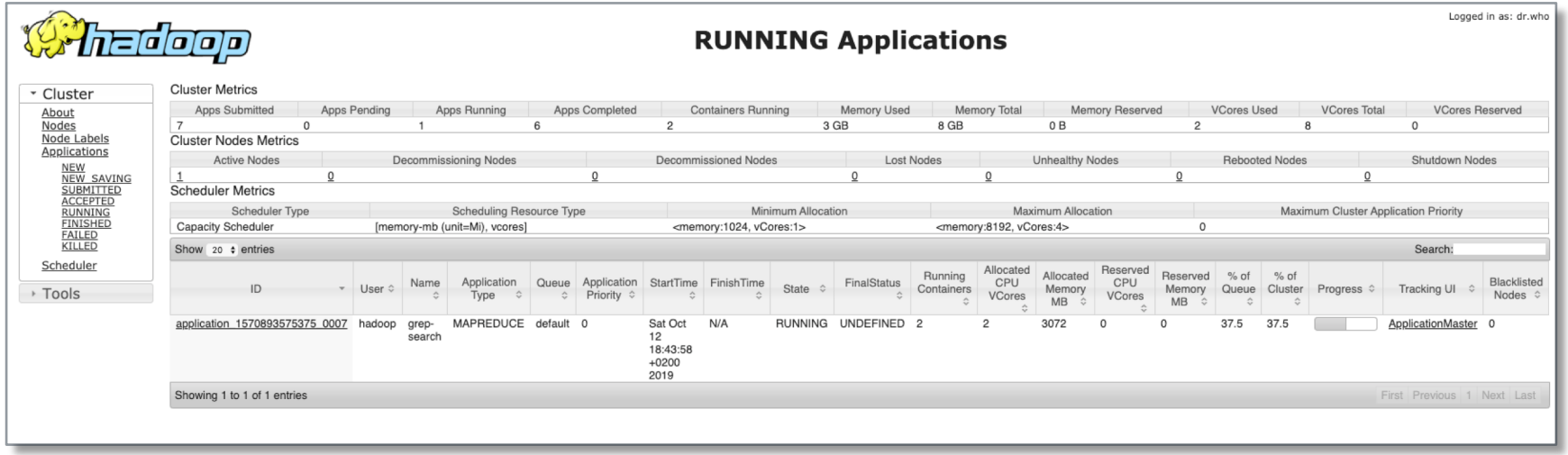
2. Use and run default MapReduce Jar (*hadoop-mapreduce-examples-3.1.2.jar*) to **grep** for string „Faust“ in text file „*Faust_1.txt*“ and count appearances of string.

```
hadoop jar hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.1.jar grep /user/hadoop/Faust_1.txt /user/hadoop/Faust_1_Count_Output 'Faust'

[…]
2019-10-12 16:44:16,517 INFO mapreduce.Job: Running job: job_1570893575375_0008
2019-10-12 16:44:27,637 INFO mapreduce.Job: Job job_1570893575375_0008 running in uber mode : false
2019-10-12 16:44:27,638 INFO mapreduce.Job:  map 0% reduce 0%
2019-10-12 16:44:31,678 INFO mapreduce.Job:  map 100% reduce 0%
2019-10-12 16:44:36,717 INFO mapreduce.Job:  map 100% reduce 100%
2019-10-12 16:44:37,735 INFO mapreduce.Job: Job job_1570893575375_0008 completed successfully
[…]
```

Solution

Exercise 3:

3. Take a look at Ressource Manager for Job Execution (http://XXX.XXX.XXX.XXX:8088/cluster/apps/RUNNING):



Solution

Exercise 2:

4. a) Copy MapReduce output file back to ubuntu local filesystem (using bash):

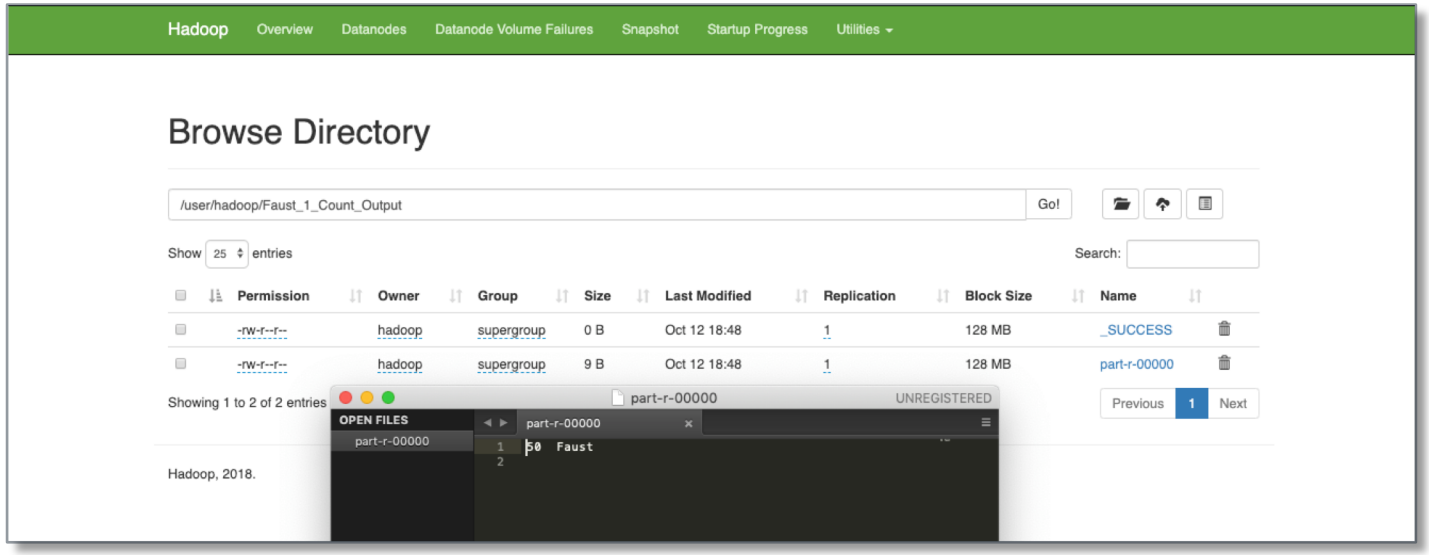
```
hadoop fs -get /user/hadoop/Faust_1_Count_Output/part-r-00000 Faust_1_Count_O
utput.csv
```

```
cat Faust_1_Count_Output.csv

50        Faust
```

Solution

Exercise 2:

4. b) Copy MapReduce output file back to ubuntu local filesystem (using Web Filebrowser):



Help

MapReduce Examples within *hadoop-mapreduce-examples-3.1.1.jar*:

aggregatewordcount:	An Aggregate based mapreduce program that counts the words in the input files.
aggregatewordhist:	An Aggregate based mapreduce program that computes the histogram of the words in the input files.
bbp:	A mapreduce program that uses Bailey-Borwein-Plouffe to compute exact digits of Pi.
dbcount:	An example job that counts the pageview logs stored in a database.
distbbp:	A mapreduce program that uses a BBP-type formula to compute exact bits of Pi.
grep:	A mapreduce program that counts the matches of a regex in the input.
join:	A job that performs a join over sorted, equally partitioned datasets.
multifilewc:	A job that counts words from several files.
pentomino:	A mapreduce tile laying program to find solutions to pentomino problems.
pi:	A mapreduce program that estimates Pi using a quasi-Monte Carlo method.
randomtextwriter:	A mapreduce program that writes 10 GB of random textual data per node.
randomwriter:	A mapreduce program that writes 10 GB of random data per node.
secondarysort:	An example defining a secondary sort to the reduce phase.
sort:	A mapreduce program that sorts the data written by the random writer.
sudoku:	A sudoku solver.
teragen:	Generate data for the terasort.
terasort:	Run the terasort.
teravalidate:	Checking results of terasort.
wordcount:	A mapreduce program that counts the words in the input files.
wordmean:	A mapreduce program that counts the average length of the words in the input files.
wordmedian:	A mapreduce program that counts the median length of the words in the input files.
wordstandarddeviation:	A mapreduce program that counts the standard deviation of the length of the words in the input files.

Solution – Exercise II

Hive, HiveQL

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 120813
movie 532594
tvEpisode 4366605
tvSeries 172627
[…]

Time taken: 36.261 seconds, Fetched: 10 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);
nm10913258 Shea Lightfoot 2019 NULL actor NULL
Time taken: 66.858 seconds, Fetched: 1 row(s)
```

Well, thats actually a bug within IMDB data:

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;

tt7286456 Joker 2019 9.0 240216
tt5813916 Dag II 2016 9.0 101524
tt1375666 Inception 2010 8.8 1880162
tt0816692 Interstellar 2014 8.6 1336209
tt4154756 Avengers: Infinity War 2018 8.5 714459
tt1675434 Intouchables 2011 8.5 692986
tt2582802 Whiplash 2014 8.5 635438
tt4154796 Avengers: Endgame 2019 8.5 575421
[…]
```


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;

39
```


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
2014 6
2009 6
2000 6
1998 5
2004 5
[…]
```


Introduction To Requirements Of Distributed Systems

(Non-)Functional Requirements Of Distributed Data-Systems

Requirements Of A Data-System

Data Storage: We need to store data and also need to be able to find it again later (*database*).

Data Querying: We need to be able to query and filter data efficiently in certain kinds of ways (*transaction* and *indices*).

Retention and Performance: We want results fast, especially of expensive read operations (*caching*).

Data Processing: We want to be able to process a huge amount of data (*batch processing*) as well as process data asynchronously (*stream processing*).

Requirements Of A Data-System

- same requirements as for the first database ***CODASYL*** (back in the 1960's)

- even though there have been a lot of databases back in time - all of them still match those same requirements.

- Evolution:
  - **relational databases** being able to handle NoSQL data (e.g. even "retirees" like *IBM DB2* or *Oracle*) as well as **NoSQL databases** being able to handle traditional SQL (e.g. ToroDB)
  - **databases** becoming message queues (e.g. *RethinkDB* or *Redis*) and the other way around **message queueing systems** become databases (e.g. Ap ache Kafka)

  → boundaries blurr

Scalability

Scalability *is the capability of data system to handle a growing amount of load (e.g. a larger amount of data needed to store or requests to handle). A data-system is considered scalable if its capable of increasing it's total through-/output under an increased load when resources (typically hardware) are added.*

- Scalability is **NOT**:
  - **a binary tag** (scalable/not-scalable), that could be attached to a data-system.

→ For any scalable data-system you need to think about:

*„If the **load** of a data system grows in a certain way, what are the options on the table for **coping with the growth**?"*

*"How can we **add ressources** (hardware) to be able to **handle the additional load**?"*

Scalability - **Load**

The ***load*** *of a data system is a measurement of the amount of computational work it performs (depending on the architecture in-place), e.g. the number of (concurrent) reads from a data storage, writes to a data storage or the ratio be tween reads and writes. The maximum load is defined by the weakest part of the architecture (=bottleneck).*

- E.g.
  - **requests/second** to a website
  - **read/write requests/second** to a data-system
  - **read/write ratio** of a data-system
  - **cash hit-rate**

Scalability - **Performance**

Performance *of a data-system is defined by system* ***throughput and response time****, e.g. number of transactions (like read/write operations), processed records (like aggregations for analytical purposes) or even system commands (like an update statistics or re-balancing of several data nodes)* ***under a given workload and for a specific time-frame****.*
It usually depends on a variety of influencable as well as uninfluencable factors of the sy stem itself, like network latency, a page fault or damaged disk.

→ The *load parameter* **increases**, but all ressources (e.g. number of server, CPU or memory) stay the same - **how is the performance of the data system affected?**

→ The *load parameter* **increases** – **how much** do you need **to increase the ressources** (e.g. number of server, CPU or memory) to keep the performance stay the same?

Scalability – **Performance Measurement**

E.g. **Throughput:**

- **read/writes per second** (in case of MongoDB up to 100.000 reads/ writes per second)
- **messages processed** (in case of Apache Kafka and Linkedin more than 2 million records per second on just 3 nodes)
- **data processed** (in case of Apache Hadoop and MapReduce terabytes of data within several seconds)

Or **Response Time:**

- **read/writes per second** (in case of MongoDB up to 100.000 reads/ writes per second)

Scalability – **Performance Measures**

Arithmetic mean:

- easy to calculate
- ignores ratios
- is highly affected by statistical outliers, so it cannot tell you how many requests, reads or writes actually have had a worse performance

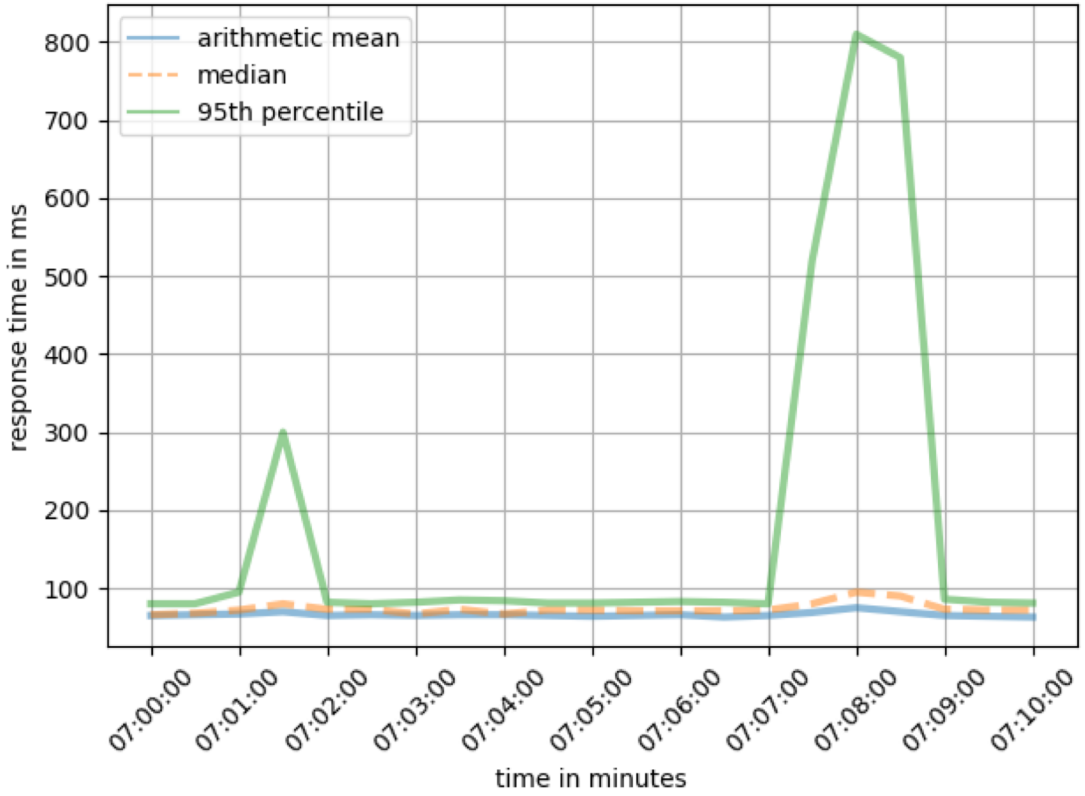
Median:

- easy to calculate
- less distorted by outliers

Percentiles (e.g. p95, p99, p999 percentiles)**:**

- easy to calculate
- not distorted by outliers

Scalability – Performance Measures



Scalability – Aproaches For Scaling

Scale Up/Vertical:

- replace a server by a more powerful one

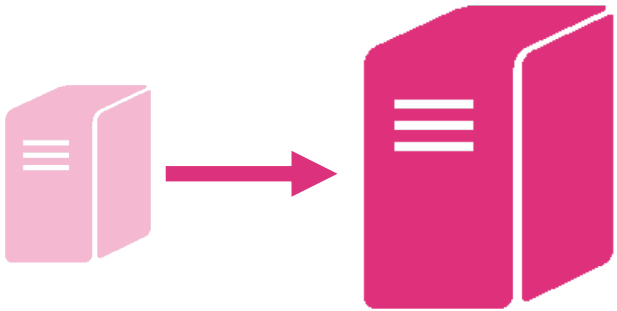
→ easier for a development

→ more expensive in terms of hard-ware

Scale Out/Horizontal:

- distribute the load towards multiple server in-stead of one

- development more complex
- cheaper in terms of hardware



→ But ist always about a mix of both

Reliability

Reliability *in terms of hardware, software or especially data-systems can be defined as the ability of a system to function as specified and expected. A reliable data-system also detects and tolerates faults due to mistakes of users, hardware or lower parts of the data-system itself as well as ensures the required performance under any expected load.*

- Or more pragmatic: "*something works correctly even if things go wrong*"

- *Typical Faults:*
  - ***Hardware** faults*
  - ***Software** faults*
  - ***Human** faults*

Reliability – Hardware Faults

- E.g.:
  - **broken HDDs** or **SSDs**
  - **faulty RAM** or **CPUs**
  - **broken power adapters**, **switches** or **whole network outages**
  - **unplugged network cables** or even connected to the wrong port
  - and many many more…

- **Seems very unlikely?**
  - imagine a 100 node Hadoop Cluster with 19 HDDs per Node = 1.900 HDDs overall
  - According to BackBlaze the average annual HDD failure rate is about 2.11%

  → approximately **each week an HDD will fail**

https://www.backblaze.com/blog/hard-drive-failure-rates-q1-2017/

Reliability – **Software Faults**

- E.g.:
  - a **runaway** and/or **zombie process** that extensively used up some shared ressource (e.g. network, CPU, RAM, disk space)
  - a **software bug** causing the whole cluster to fail (e.g. the Hadoop Ressource Manager YARN once had a bug, that if you removed a cgroup under some circumstances (*race conditions*) a *kernel panic* and in this way a failure of multi-ple server was caused
  - **bottleneck**: a service the whole data-system depends on slows done, becomes unresponsive or fails
  - **cascading failures** (e.g. one server of the data-system fails due to heavy network traffic, causing the other servers to take over
    → in this way increasing network traffic for them too and finally all server will fail)

https://issues.apache.org/jira/browse/YARN-2809

Reliability – Human Faults

- Most unreliable factor: **humans**

- Approaches for making a data-system reliable, even in terms of **humans**:

  - decouple places where people make the **most failures** from **places they can cause failures**, e.g. using production and development environments or providing interfaces or frameworks for an API instead of direct API access
  - **use extensive testing** (e.g. unit test, system tests, integration tests) and automize them
  - **measuring and monitoring** (e.g. performance metrics, error rates) all ows to check wether assumptions or constraints are violated at an earl y stage

Maintainability

- **Maintenance** = one of the biggest costs in software development
- A data-system should be designed to minimize maintenance effort

- 3 Main principles to follow:

  - **1. Operability:** *make it easy to run the data-system*
    - **good documentation** and **operational model** (a data-system which can be understand easily can be operated more easily)
    - **transparency** (visibility into the data system and runtime behaviour, e.g. by log files or monitoring tools)
    - **no dependencies** between single services or server (allow single server to go down for maintenance tasks, e.g. patches, update or restarts)
    - **self-healing** if possible, but also possibilities to override for operators

Maintainability

- **2. Simplicity:** *make it easy to understand the data-system*
  - make use of **abstraction** and **reduce complexity** (less complex doesn't require reducing functionaility, it's more about removing unnecessary complexity)
  - **clearly defined interfaces**
  - **no over-engineering**

- 3. **Evolvability:** *make it easy to adapt/change the data-system*
  - **continues integration**
  - **test-driven development**
  - **pair-programming**
  - …

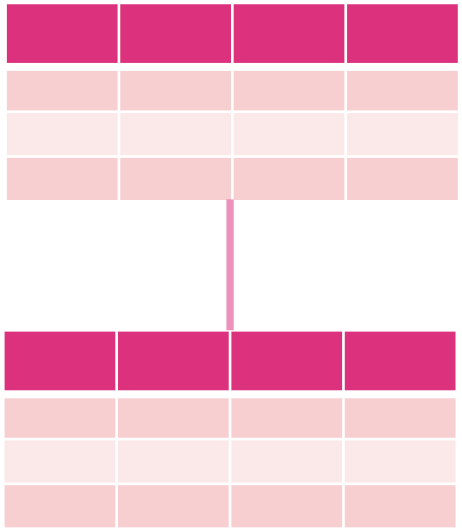
Introduction To Data Models And Access

Relational and Non-Relational Data Models, Data Access

Data Models (Relational/Non-Relational)

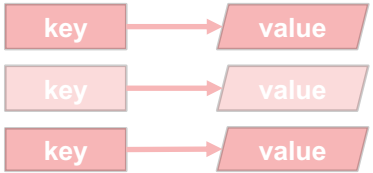
RELATIONAL

Row/Column Based

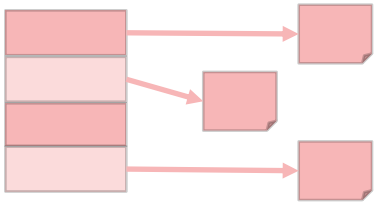


NON-RELATIONAL

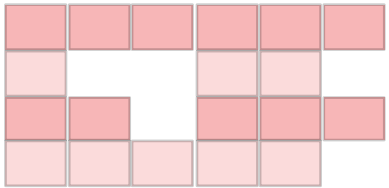
Key-Value



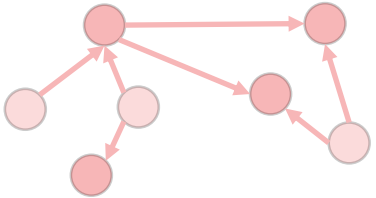
Document



Column Family



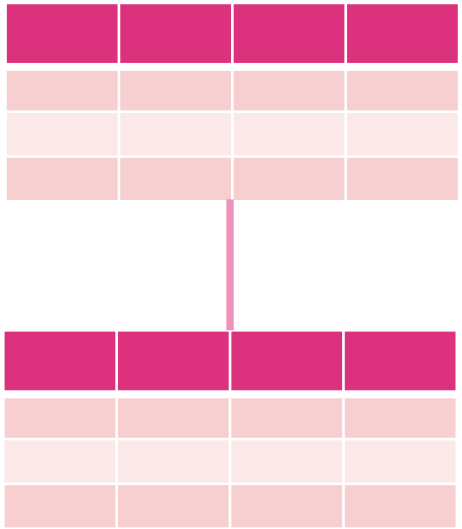
Graph



Relational Data Model

RELATIONAL

Row/Column Based



NON-RELATIONAL

Key-Value

Document

Column Family

Graph

Relational Data Model

- Originally introduced by **Edgar Frank Codd** in 1970

*The **relational model** is an approach to managing data using a structure and language consistent with first-order predicate logic where all data is represented in terms of tuples and grouped into relations.*

- Idea of: hide implementation details (representation of data in data store)
- By providing: a *declarative* and *read-on-schema* interface

→ Developers/user can easily state what information the database contains and what information they want
→ The database will take care of describing, storing and retrieving data

Relational Data Model - Example

- **Example**: Facebook Profile Page as relational model

- table `user` = main entity
  - stores *primary key* (`id`)
  - stores basic information (e.g. `first_name`, `last_name`)

- As people may have:
  - worked in multiple companies
  - studied at multiple universities
  - lived at different cities

  → separate tables with *foreign key* (`user_id`):
    - `companies`
    - `universities`
    - `cities`

table: ***user***

id	first_name	last_name	is_verified	married_to	...
1	Mark	Zuckerberg	true	4711	
4711	Priscilla	Chan	true	1	

table: ***universities***

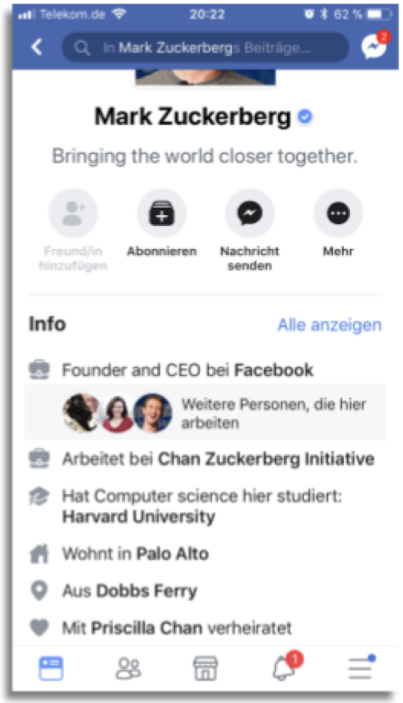
id	user_id	university_name	subject	...
1378	1	Harvard University	Computer Science	
1379	4711	University of California	Medicin	

table: ***companies***

id	user_id	company_name	...
1337	1	Facebook	
1338	1	Chan Zuckerberg Initiative	
1339	4711	UCSF Benioff Children's Hospital	

table: ***cities***

id	user_id	city_name	status	...
1378	1	Palo Alto	living	
1379	1	Dobbs Ferry	born	
1379	4711	Palo Alto	living	



Relational Data Model – List Of Software

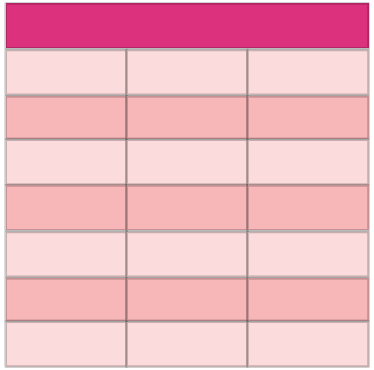
List of Software [edit]

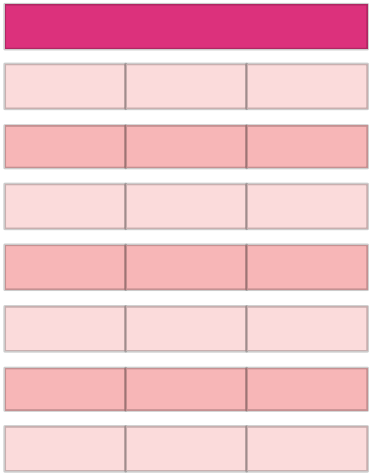
- 4th Dimension
- Adabas D
- Alpha Five
- Apache Derby
- Aster Data
- Amazon Aurora
- Altibase
- CA Datacom
- CA IDMS
- Clarion
- ClickHouse
- Clustrix
- CSQL
- CUBRID
- DataEase
- Database Management Library
- Dataphor
- dBase
- Derby aka Java DB
- Empress Embedded Database
- EXASolution
- EnterpriseDB
- eXtremeDB
- FileMaker Pro
- Firebird
- FrontBase
- Google Fusion Tables
- Greenplum
- GroveSite
- H2
- Helix database
- HSQLDB
- IBM DB2
- IBM Lotus Approach
- IBM DB2 Express-C
- Infobright
- Informix
- Ingres
- InterBase
- InterSystems Caché
- LibreOffice Base
- Linter
- MariaDB
- MaxDB
- MemSQL
- Microsoft Access
- Microsoft Jet Database Engine (part of Microsoft Access)
- Microsoft SQL Server
- Microsoft SQL Server Express
- SQL Azure (Cloud SQL Server)
- Microsoft Visual FoxPro
- Mimer SQL
- MonetDB
- mSQL
- MySQL
- Netezza
- NexusDB
- NonStop SQL
- NuoDB
- Omnis Studio
- Openbase
- OpenLink Virtuoso (Open Source Edition)
- OpenLink Virtuoso Universal Server
- OpenOffice.org Base
- Oracle
- Oracle Rdb for OpenVMS
- Panorama
- Pervasive PSQL
- Polyhedra
- PostgreSQL
- Postgres Plus Advanced Server
- Progress Software
- RDM Embedded
- RDM Server
- R:Base
- SAND CDBMS
- SAP HANA
- SAP Adaptive Server Enterprise
- SAP IQ (formerly known as Sybase IQ)
- SQL Anywhere (formerly known as Sybase Adaptive Server Anywhere and Watcom SQL)
- solidDB
- SQLBase
- SQLite
- SQream DB
- SAP Advantage Database Server (formerly known as Sybase Advantage Database Server)
- Teradata
- Tibero
- TimesTen
- Trafodion
- txtSQL
- Unisys RDMS 2200
- UniData
- UniVerse
- Vectorwise
- Vertica
- VoltDB

https://en.wikipedia.org/wiki/List_of_relational_database_management_systems

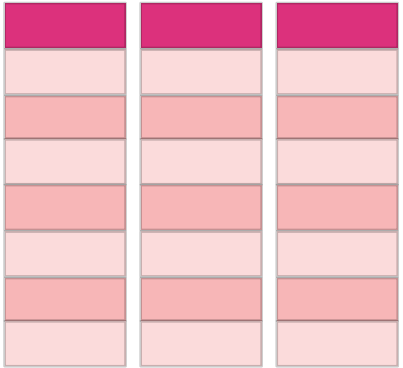
Relational Data Model – **Row vs. Column-Based**

Row-Based:
→ rows are stored continuesly
→ e.g. Oracle, IBM DB2, Microsoft SQL Server, MySQL
→ some also support column-based





Column-Based:
→ columns are stored continuesly
→ e.g. Hbase, Parquet, SAP HANA, Teradata



Relational Data Model – Row vs. Column-Based

Operation	Row-Based	Column-Based
Compression	**Low**	**High**
Column Scans	**Slow** (Multiple Reads)	**Fast** (One Read)
Insert Of Records	**Fast** (One Insert)	**Slow** (Multiple Inserts)
Single Record Queries	**Fast** (One Read)	**Slow** (Multiple Reads)
Single Column Aggregation	**Slow** (Full Table Scan)	**Fast** (Only Column Scan)
Typical Use Cases	**Transactional**	**Analytical**

Relational Data Model - **SQL**

- Originally introduced as „SEQUEL“ by **Donald D. Chamberlin** and **Raymond F Boyce** in 1974 and inspired by **Edgar Frank Codds** relational data model in 1970
- ANSI Standard 1958, ISO 1987

- → todays **most widely** used **database language**

- → **even by NoSQL** data-systems, e.g.:
  - Apache Cassandra (*CQL*)
  - Apache Hadoop (*HiveQL*)
  - Apache Flink (*FQL*)

Relational Data Model - SQL

Language:

- SQL = *declarative*
  - → no need to worry about how the data is stored and retrieved
  - → automatic performance optimization *(query optimizer)*
- *imperative* = e.g. MapReduce, Spark, …
  - → need to think about data storage
  - → no query optimizer

Usual DML Statements:

CREATE/DROP/ALTER TABLE…
INSERT INTO … VALUES…
UPDATE … SET … WHERE…
DELETE FROM .. WHERE…
…

Structure:

```
SELECT      -- attributes
FROM        -- tables
WHERE       -- conditions
GROUP BY    -- grouping attributes
HAVING      -- grouping conditions
ORDER BY    -- attributes
```

Keywords:

DISTINCT, LIMIT, AS
MIN, MAX, AVG, COUNT, SUM
AND, OR, NOT, IN, LIKE, ANY
UNION, JOIN
ASC, DESC
…

Relational Data Model – SQL Example

Query:

```
select m.tconst, m.original_title, m.start_year, r.average_rating, r.num_votes
from imdb_movies m JOIN imdb_ratings r on (m.tconst = r.tconst) WHERE r.average_rating > 6
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 LIMIT 200
```

Result:

	ABC m.tconst	ABC m.original_title	123 m.start_year	123 r.average_rating	123 r.num_votes
1	tt0816692	Interstellar	2.014	8,6	1.213.141
2	tt4154756	Avengers: Infinity War	2.018	8,6	492.952
3	tt1675434	Intouchables	2.011	8,5	635.669
4	tt2582802	Whiplash	2.014	8,5	571.172
5	tt5074352	Dangal	2.016	8,5	105.207
6	tt1345836	The Dark Knight Rises	2.012	8,4	1.331.811
7	tt1853728	Django Unchained	2.012	8,4	1.153.183
8	tt2380307	Coco	2.017	8,4	219.957
9	tt5311514	Kimi no na wa.	2.016	8,4	108.553
10	tt2106476	Jagten	2.012	8,3	226.819
11	tt1832382	Jodaeiye Nader az Simin	2.011	8,3	186.217
12	tt0993846	The Wolf of Wall Street	2.013	8,2	976.551
13	tt2096673	Inside Out	2.015	8,2	499.721
14	tt1291584	Warrior	2.011	8,2	389.935
15	tt3170832	Room	2.015	8,2	288.153
16	tt5027774	Three Billboards Outside Ebbing, Missouri	2.017	8,2	280.790

Relational Data Model - SQL

- **Strenghts:**
  - Consistency → ACID
  - Universal → a lot of data types, linked and unlinked data, “Independance“ of RDBS
  - Strict Schema → Data Quality (*Garbage-In – Garbage-Out*), Error Prevention, Compression

- **Weaknesses:**
  - Strict Schema:
    → needs to be altered at any data format change
    → data needs to be migrated
  - Object-Relational-Impedance Mismatch:
    - E.g. objects, structs, ...
    - needs ORMs
    - usually slows and complicates data access

Relational Data Model – **Advantages/Disadvantages**

- **Data Fetching:** rows can easily be inserted and fetched all-at-once by using an `id` or a *primary/foreign key,* unlike e.g. document-oriented data models, where you usually need to:
  - make use of ***access paths***
  - need to think about **nested structures**
  - need to worry about **unknown fields** as those systems are usually *schema-on-read*
  - or intensively need to **think about possible performance issues** and how the system will probably execute your query as the execution engine is usually not that mature as the one of a relational system
  - it's more **difficult to understand how the systems works**, as sometimes you don't even have a *query-explain* feature like in relational databases, so you won't be able to investigate or guess in advance how it will behave in detail during execution.

Relational Data Model – **Advantages/Disadvantages**

- **Object-Orientation:**
  applications need *translation layer* for data-system: **ORM Frameworks,** like e.g.*:*
  - ***Hibernate*** in case of Java
  - ***SQLAlchemy*** in case of Python

- **Many-to-many relations**:
  - **relational data model**: foreign key constraints
    → inexpensive (indices etc.)
  - **NoSQL data model**: nested structures, document references
    → expensive IO/CPU operations

- **Writes/Updates**:
  - **relational data model**: efficient, as single rows will be updated
  - **NoSQL data model**: usually requires rewriting a whole document

Relational Data Model – **Advantages/Disadvantages**

- **Constraints:**
  - **relational data model**:
    - Data Types
    - Primary/Foreign Keys
    - …

    → highly serves **data integrity**
  - **NoSQL data model**:
    - No or less constraints

    → Freedom but **vulnerable** to *garbage-in – garbage-out*

- **Operations**:
  - **relational data model**: relational algebra

    → mighty language
  - **NoSQL data model**: data-system specific

    → usually less mighty

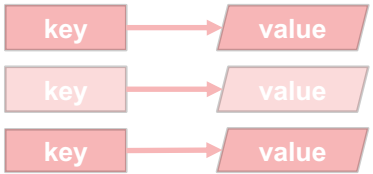
Non-Relational Data Model

RELATIONAL

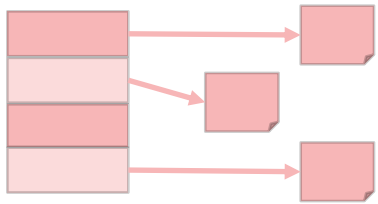
Row/Column Based

NON-RELATIONAL

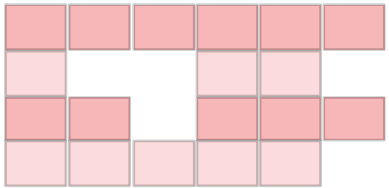
Key-Value



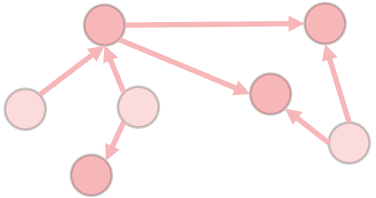
Document



Column Family



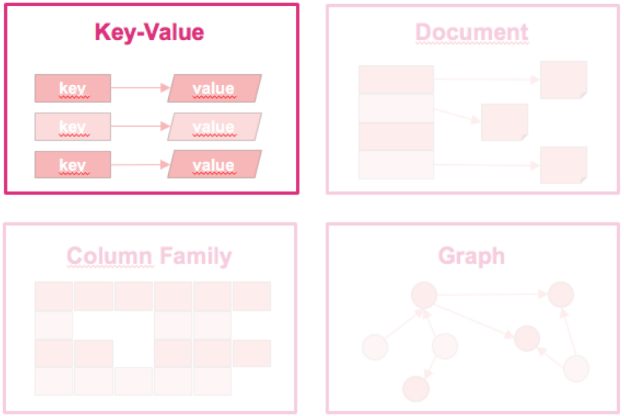
Graph



Non-Relational Data Model

NoSQL in terms of BigData, is a theorem for managing data using document-oriented, key-value or graph structures, where all data is represented in terms of documents, key-value pairs, graphs (nodes, edges, properties) or mixed approaches.

Non-Relational Data Model – **Key-Value**



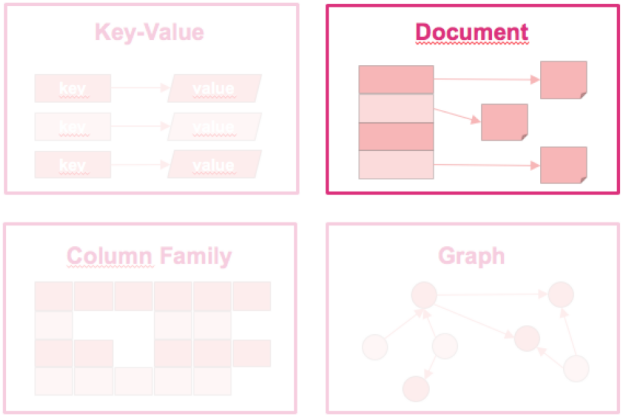
- **Examples:**
  - Redis,
  - BerkeleyDB,
  - VoldemortDB,
  - ArangoDB,
  - Riak, …

- **Strenghts:**
  - **fast queries** (value lookups)
  - **fast inserts** (key-value pairs)
  - **easy to replicate** and **distribute** (e.g. *consistent hashing*)

- **Weaknesses**:
  - **no** or **less efficient** and slow:
    - aggregation
    - filtering
    - joining

    → needs to be done by application

Non-Relational Data Model – **Document**



- **Examples:**
  - MongoDB,
  - Couchbase,
  - RethinkDB,
  - ArangoDB,
  - DynamoDB,
  - CouchDB, …

- **Strenghts:**
  - **schema flexibility** (documents can have different f ormats)
  - **fast inserts** and **point queries**
  - **data locality**
  - **easy to replicate** and **distribute** (e.g. *to achieve l oad balancing and failure tolerance*)
  - **application code simplicity**

- **Weaknesses**:
  - **updates** on documents are usual **less efficient/ IO expensive** (especially if size changes)
  - **no** or **less efficient** and slow:
    - aggregation
    - filtering
    - joining

Non-Relational Data Model – **Document**

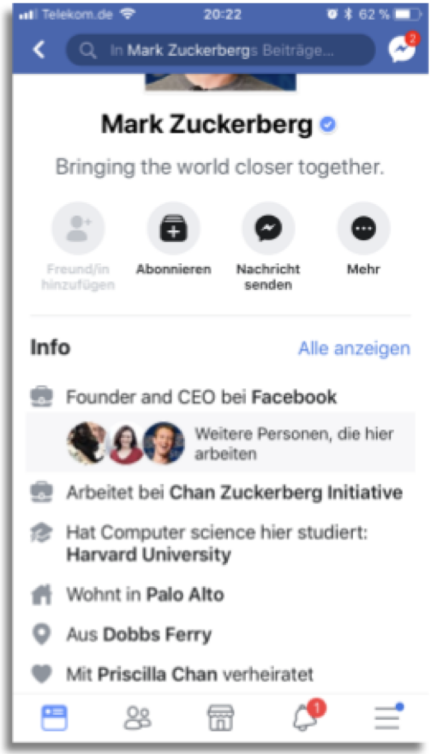
- **Example**: Facebook Profile Page as document model

- No need to fetch mutliple tables (***data locality***)

- **Easy to parse** by applications (*object-oriented*)

- schema flexibility (no expensive ALTER TABLE … needed)

```
{
  "id": 1,
  "first_name": "Mark",
  "last_name": "Zuckerberg",
  "is_verified": "true",
  "married_to": 4711,
  "universities": [
    {
      "university_name": "Harvard University",
      "subject": "Computer Science"
    }
  ],
  "companies": [
    {
      "company_name": "Facebook"
    },
    {
      "company_name": "Chan Zuckerberg Initiative"
    }
  ],
  "cities": [
    {
      "city_name": "Palo Alto",
      "status": "living"
    },
    {
      "city_name": "Dobbs Ferry",
      "status": "born"
    }
  ]
}
```



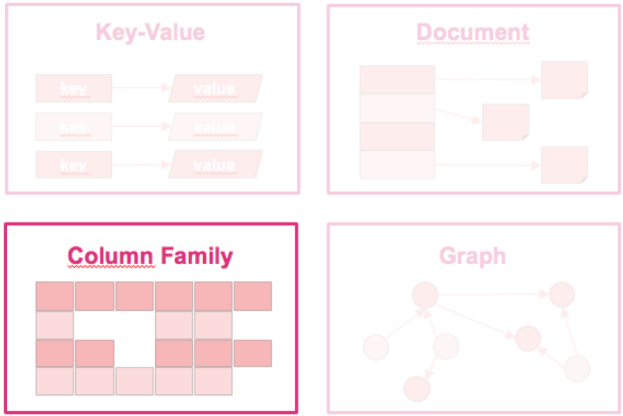
Non-Relational Data Model – **Document**

SQL	MongoDB
SELECT * **FROM** user	`db.user.find()`

SELECT id, first_name, last_name
FROM user **WHERE** is_verified = true
	`db.user.find(`
	`{ is_verified : true },`
	`{ id : 1, first_name: 1,`
	`last_name: 1 }`
	`)`

INSERT INTO user(id, first_name, last_name
VALUES (1, „Mark“, „Zuckerberg“)
	`db.user.insertOne(`
	`{ id : 1,`
	`first_name: “Mark“,`
	`last_name: “Zuckerberg“ }`
	`)`

Non-Relational Data Model – **Column Family**

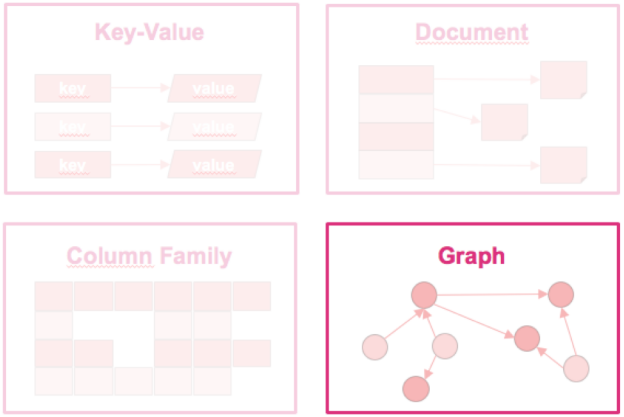


- **Examples:**
  - Cassandra,
  - BigTable
  - HBase

- **Strenghts:**
  - **schema flexibility** (documents can have different f ormats)
  - **fast inserts** and **point queries**
  - **Fast point queries/value lookups**
  - **easy to replicate** and **distribute** (e.g. *to achieve l oad balancing and failure tolerance*)

- **Weaknesses**:
  - **non-point queries** are **less efficient**/ **IO expensive** (e.g. range queries)
  - **no** or **less efficient** and slow:
    - aggregation
    - filtering
    - joining

Non-Relational Data Model – **Graph**



- **Examples:**
  - Neo4j,
  - ArangoDB,
  - OrientDB,
  - Titan,
  - Giraph, …

- **Strenghts:**
  - **many-to-many relationships** (unlike all other data models
  - **schema flexibility** (due to edges and property definition)
  - **efficient relationship queries**
  - usually support **ACID**

- **Weaknesses**:
  - not useful for transactional data or CRUD operations

Non-Relational Data Model – **MapReduce**

- **MapReduce** = **programming paradigm** and an associated **implementation** for **processing** and **generating large datasets in parallel** and **distributed** on a cluster
- originally introduced by Jeffrey Dean and Sanjay Ghemawat (Google Inc.) in 2004

- MapReduce is neither a declarative language nor a an imperative programming language → it's something in between

- The paradigm is based on specifying:
  - a **map function**, which performs filtering and sorting, resulting in an intermediate set of key/value pairs and
  - a **reduce function** that merges all intermediate values associated with the same key (e.g. sum all values).

- MapReduce Jobs are automatically parallized and executed on several nodes of a cluster

MapReduce

- The MapReduce **runtime system** (e.g. YARN in case of Hadoop) takes care of:
  - the details of providing and partitioning the input data,
  - scheduling the application code execution across all cluster nodes,
  - saving intermediate states,
  - handling node failures,
  - inter-node communication and much more.

MapReduce – **WordCount**

```
1  map(String key, String value):
2    // key: document name
3    // value: document content
4    for each word w in value:
5      EmitIntermediate(w, 1);
6
7  reduce(String key, Iterator values):
8    // key: a word
9    // values: a list of counts
10   int result = 0;
11   for each v in values:
12     result += v;
13   Emit(key, result);
```

The **map function** **takes** one **pair of data** with a type in one data dom ain, and **returns** a **list of pairs** in a different domain:

$$map(k1,v1) \rightarrow list(k2, v2)$$

This produces a list of pairs (keyed by k2) for each call.

After that, the MapReduce framework collects all pairs with the same key (*k2*) from all lists and groups them together, creating one group for each key.

The **reduce function** runs in parallel for each group, which in turn produces a collection of values (*v3*) to an associated key (*k2*) within the same domain:

$$reduce(k2, list(v2)) \rightarrow list(k2, v3)$$

The returns of all reduce processes are collected as the desired result list.

MapReduce – **WordCount**

```
document1 = "Da steh ich nun, ich armer Tor!";
document2 = "Und bin so klug als wie zuvor;";
document3 = "Heiße Magister, heiße Doktor gar";
document4 = "Und ziehe schon an die zehen Jahr";
document5 = "Herauf, herab und quer und krumm":
document6 = "Meine Schüler an der Nase herum.";
```

Code Snippet 2.2: MR Example - *Word Count (Input Documents)*

```
map(document1, "da steh ich nun ich armer tor");
map(document2, "und bin so klug als wie zuvor");
map(document3, "heiße magister heiße doktor gar");
map(document4, "und ziehe schon an die zehen jahr");
map(document5, "herauf, herab und quer und krumm");
map(document6, "meine schüler an der nase herum");
```

Code Snippet 2.3: MR Example - *Word Count (`map()` Calls)*

```
p1 = [ ("da",1), ("steh",1), ("ich",1), ("nun",1), ("ich",1),
       ("armer",1), ("tor",1) ];
p2 = [ ("und",1), ("bin",1), ("so",1), ("klug",1), ("als",1),
       ("wie",1), ("zuvor",1) ];
...
p6 = [ ("meine",1), ("schüler",1), ("an",1), ("der",1), ("nase",1),
       ("herum",1)];
```

Code Snippet 2.4: MR Example - *Word Count (Partial `map()` Results)*

MapReduce – **WordCount**

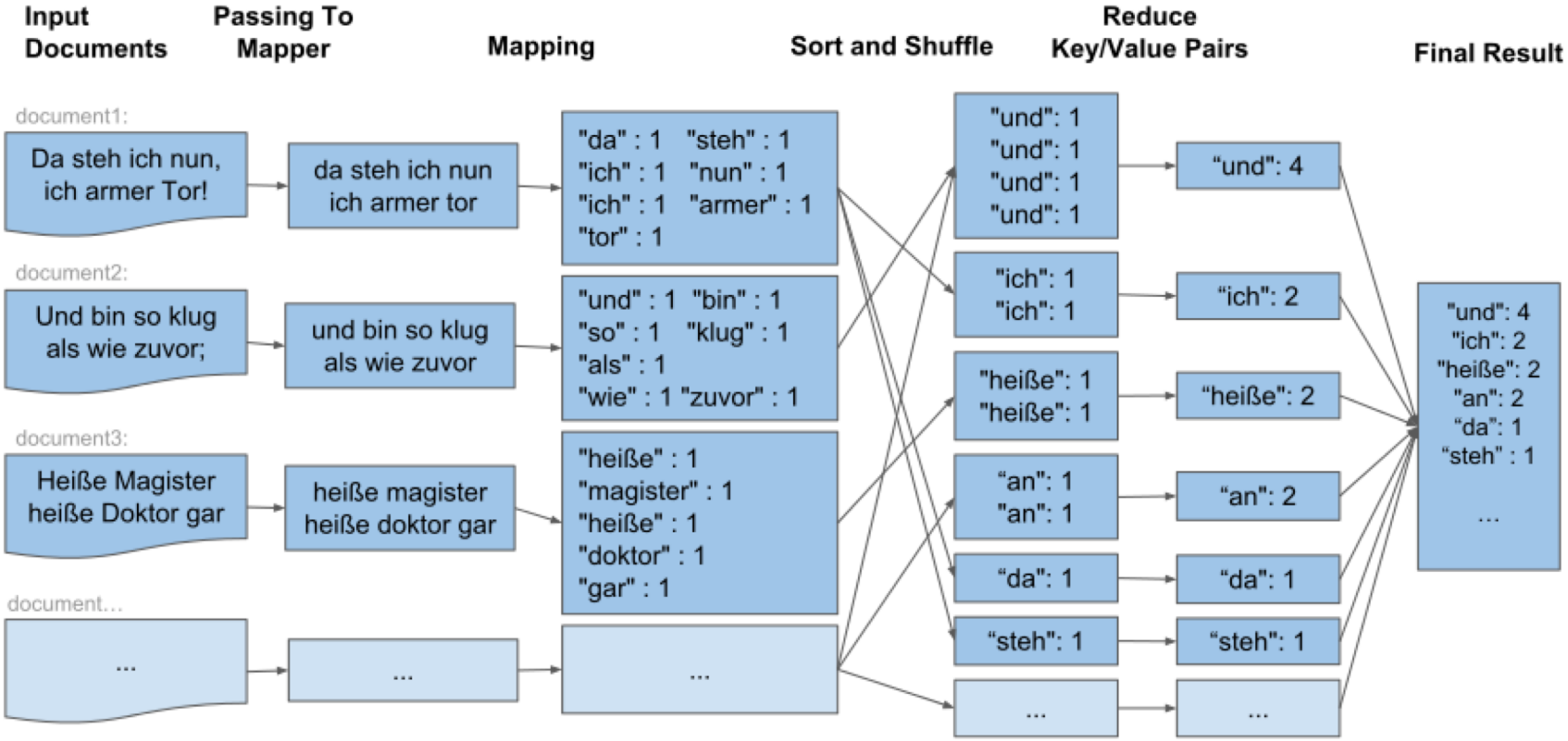
```
reduce("da", [1]); // = ("da", 1)
reduce("ich", [1,1]); // = ("ich", 2)
reduce("und", [1,1,1,1]); // = ("und", 4)
...
```

Code Snippet 2.5: MR Example - *Word Count (`reduce()` Calls)*

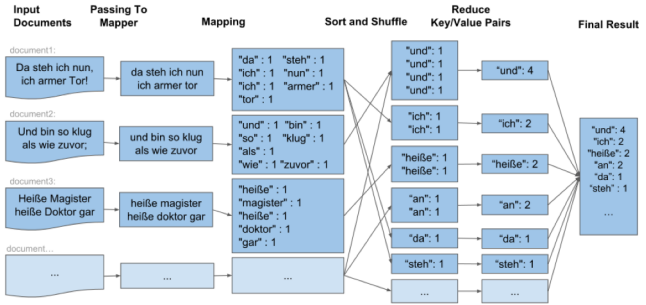
```
result = [ ("und", 4), ("ich",2), ("heiße",2), ("an", 2), ("da",1),
    ("steh",1), ("nun",1), ("armer",1), ("tor",1), ("bin",1), ("so",1),
    ("klug",1). ("als",1), ("wie",1), ("zuvor",1), ("magister",1),
    ("doktor",1), ("gar",1), ("ziehe",1), ("schon",1), ("die",1),
    ("zehen",1), ("jahr",1), ("herauf",1), ("herab",1), ("quer",1),
    ("krumm",1), ("meine",1), ("schüler",1), ("der",1), ("nase",1),
    ("herum",1) ]
```

Code Snippet 2.6: MR Example - *Word Count (Final Result)*

MapReduce – **Phases**



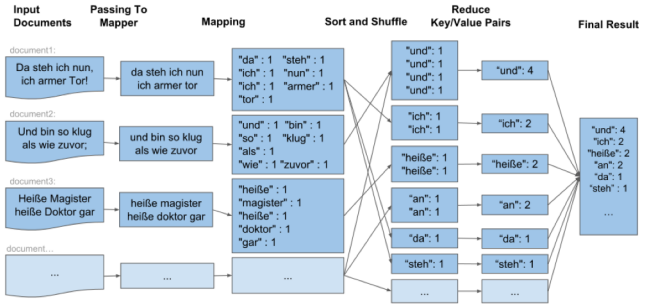
MapReduce – Phases



1. Map Phase: Each map function gets multiple key/value pairs and processes each of them separately. All Map processes run independently of each other, faciliating the whole processing to run concurrently on multiple nodes. In our example resulting in to several lists of words with an associated count of one and represented as key/values pairs.

2. Sort and Shuffle Phase: Phase in between Map and Reduce with the purpose of transferring data from map to reduce processes efficiently, which is usually done automatically by the MapReduce framework.
All output produced by the map processes is grouped and sorted by their key, partioned and distributed to the reduce processes.
A common way of partitioning is to hash the keys and use the hash value modulo amount of reduce processes to achieve an evenly distribution of load among all nodes of a cluster (the total number of partitions is equal to the number of reduce processes). Usually you do not need to make use of the frameworks default Shuffle, Sort and Partitioning, for instance using Hadoop you could implement your own ones, but as those parts are centerpieces in case of the whole MapReduce performance, think twice about not using the default ones.

MapReduce – Phases



3. Reduce Phase: The reduce function gets called once for each unique key. All Reduce processes run indepently of each other, faciliating the whole processing to run concurrently on multiple nodes. In this way the reduce functions iterates through all values associated to a key and produces zero or more output.
In case of the WordCount example an increment happens for each *value* (count) associated to the same *key* (word).

4. Output Phase: The output of all reduce processes are collected and consolidated by the MapRedcue framework and usually written to a distributed file system.

Beside the listed phases, most MapReduce frameworks also make use of a **phase** called **combiner**, which happens between Map phase and Sort&Shuffle phase. The combiner is also known as a "*mini-reducer*", it takes the intermediate output of the map processes - shuffles, sorts and reduces them partly by combining key/value pairs with the same key.
In case of our WordCount example (Figure 2.8 on page 52) the output of the first mapper wouldn't be:

```
[ ("da",1), ("steh",1), ("ich",1), ("nun",1), ("ich",1), ("armer"
   ,1), ("tor",1) ];
```

but

```
[ ("da",1), ("steh",1), ("ich",2), ("nun",1), ("armer",1), ("tor"
   ,1) ];
```

HandsOn – **MapReduce**

Quick Introduction To Java and MapReduce

WordCount – Mapper

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
```


WordCount – Reducer

```
  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```


Run Java MapReduce Job

1. Compile Java file:

```
hadoop/bin/hadoop com.sun.tools.javac.Main WordCount.java
```

2. Create Jar File:

```
jar cf WordCount.jar WordCount*.class
```

3. Execute MapRedcue Job:

```
hadoop jar WordCount.jar WordCount /user/hadoop/Faust_1.txt
/user/hadoop/wordcount_output
```


Run Java MapReduce Job

4. Take a look at the results:

```
hadoop fs -cat /user/hadoop/wordcount_output/part-r-00000 | head -10
"Allein,  1
"Alles    1
"Als      1
"Der      1
"Die      2
"Er       2
"Ich      4
"Im       1
"Mein     1
"Nur      1
[…]
```


HandsOn – **HiveServer2**

Quick Introduction To HiveServer2

Exercises III 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:

```
hive/bin/hiveserver2

2018-10-02 16:19:08: 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 = b8d1efb3-fc8c-4ec8-bdf0-6a9a41e2ddaa
Hive Session ID = 32503981-a5fd-497e-b887-faf3ec1e686e
Hive Session ID = 00f7eab4-5a29-4ce4-ad97-e90904d9206f
Hive Session ID = 100e54c5-14c6-4acc-b398-040152b08ebf
[…]
```


Connect To HiveServer2 via JDBC

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

Mac OSX:

```
wget https://dbeaver.io/files/dbeaver-ce-latest-installer.pkg
```

Linux (Debian):

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

Linux (RPM):

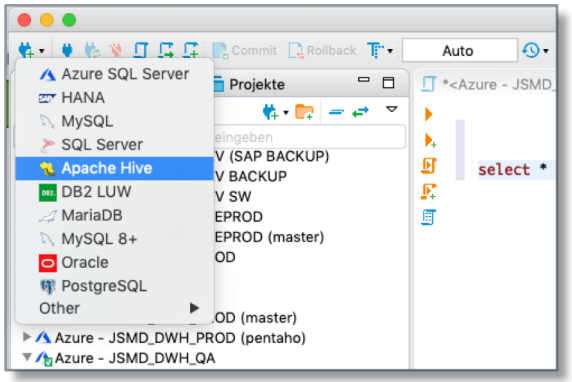
```
wget https://dbeaver.io/files/dbeaver-ce-latest-stable.x86_64.rpm
```

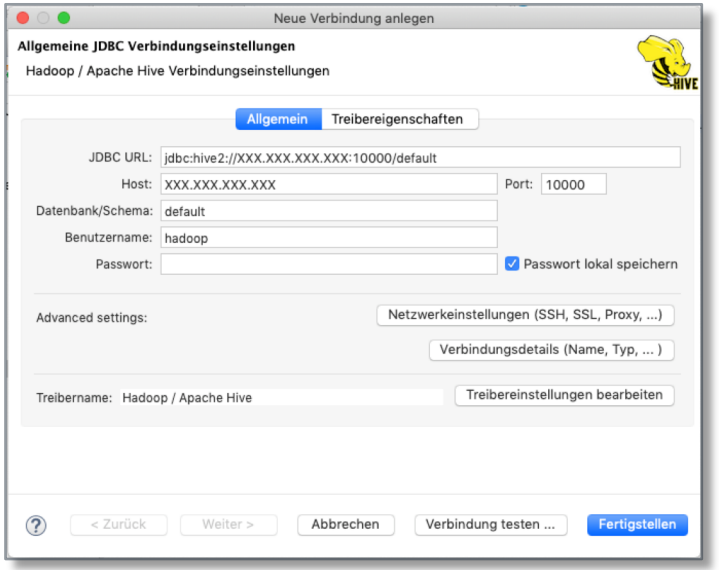
Windows:

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


Connect To HiveServer2 via JDBC

2. Configure Connection To Hive Server:





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 them into HDFS:

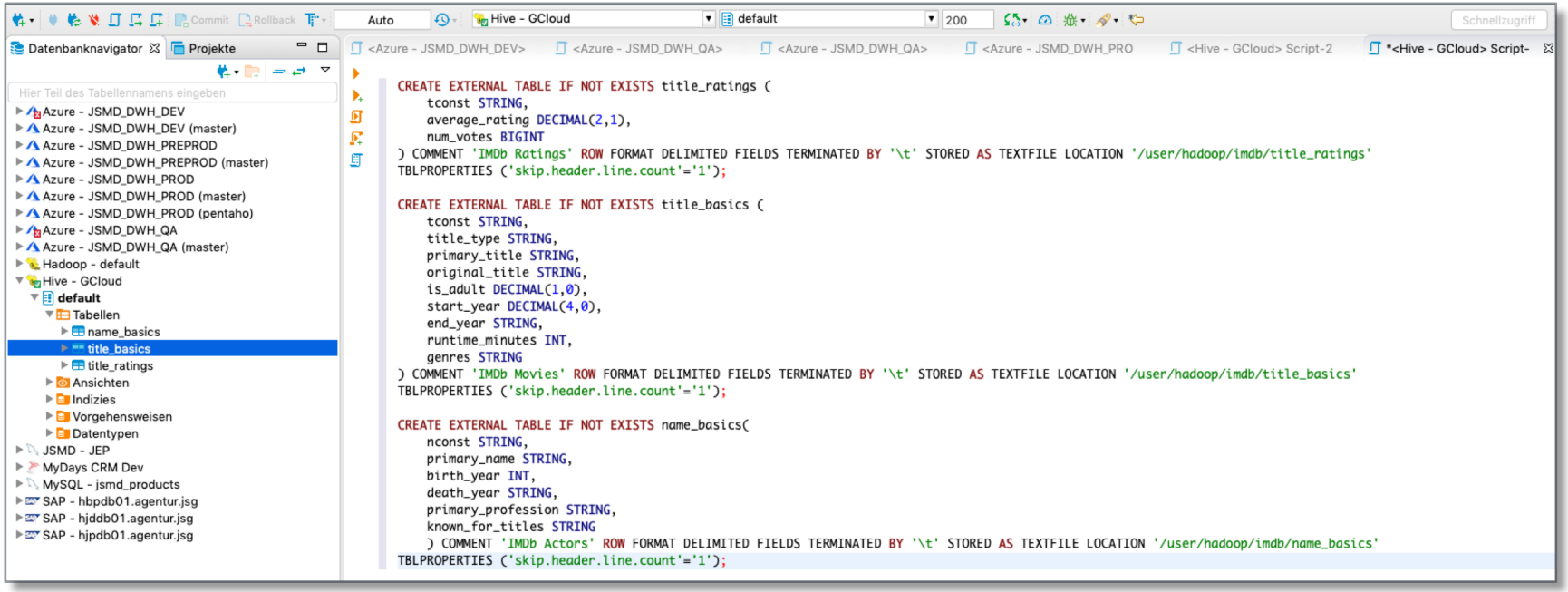
```
hadoop fs -mkdir /user/hadoop/imdb
```

```
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:

```
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;
```

Result

SELECT m.tconst, m.original_title, m.start_year, r.a | Geben Sie einen SQL-Ausdruck ein, um die Ergebnis

Gitter | Text

	tconst	original_title	start_year	average_rating	num_votes
1	tt5813916	Dag II	2.016	9,0	101.581
2	tt7286456	Joker	2.019	8,9	331.283
3	tt1375666	Inception	2.010	8,8	1.883.414
4	tt0816692	Interstellar	2.014	8,6	1.339.327
5	tt4154756	Avengers: Infinity War	2.018	8,5	717.106
6	tt1675434	Intouchables	2.011	8,5	694.378
7	tt2582802	Whiplash	2.014	8,5	637.001
8	tt4154796	Avengers: Endgame	2.019	8,5	582.173
9	tt1345836	The Dark Knight Rises	2.012	8,4	1.414.773
10	tt1853728	Django Unchained	2.012	8,4	1.243.114

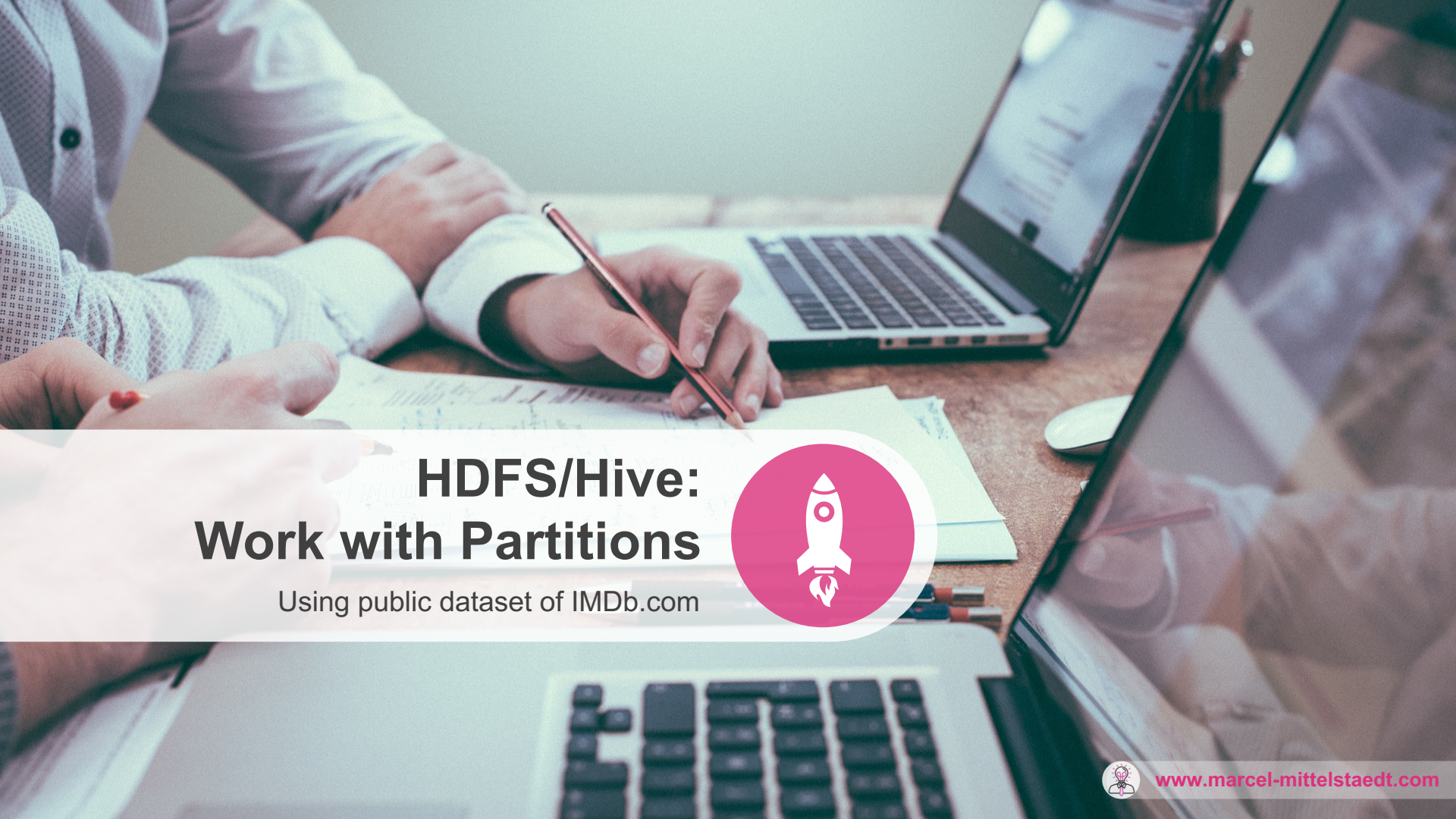
Exercises III

HiveServer2 For Remote Connections

HiveServer2 Exercises

1. Execute Tasks of previous HandsOn Slides.

HandsOn – HDFS/Hive Partitioning and HiveServer2

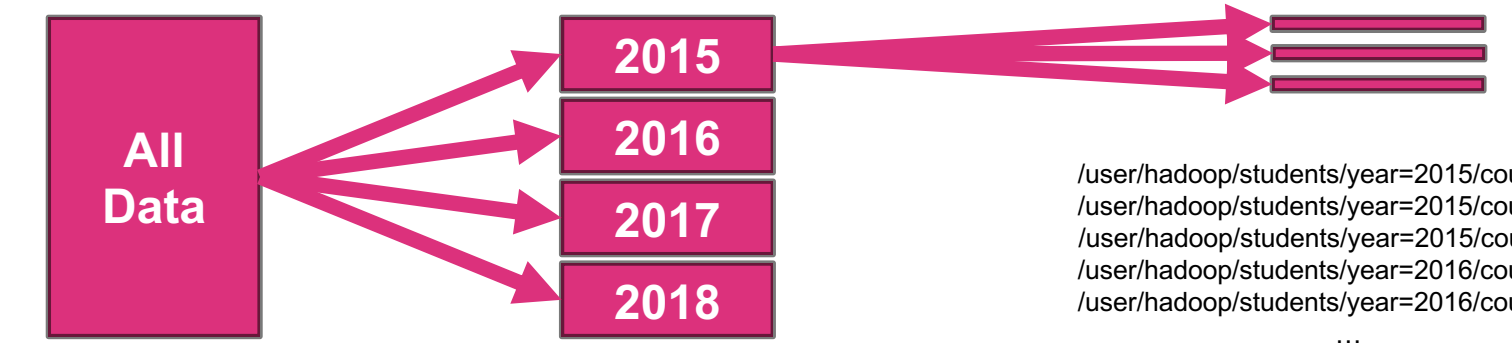


HDFS/Hive: Work with Partitions

Using public dataset of IMDb.com

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**



Static Partitioning – **Create Partitioned Table**

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

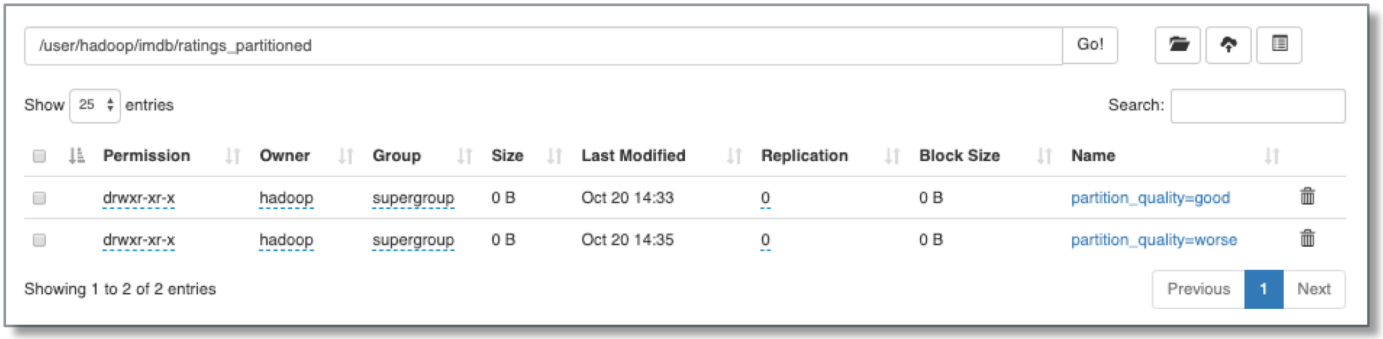
```
CREATE TABLE IF NOT EXISTS title_ratings_partitioned (
    tconst STRING,
    average_rating DECIMAL(2,1),
    num_votes BIGINT
) PARTITIONED BY (partition_quality STRING)
STORED AS ORCFILE LOCATION '/user/hadoop/imdb/ratings_partitioned';
```


Static Partitioning – INSERT Into Table via Hive

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

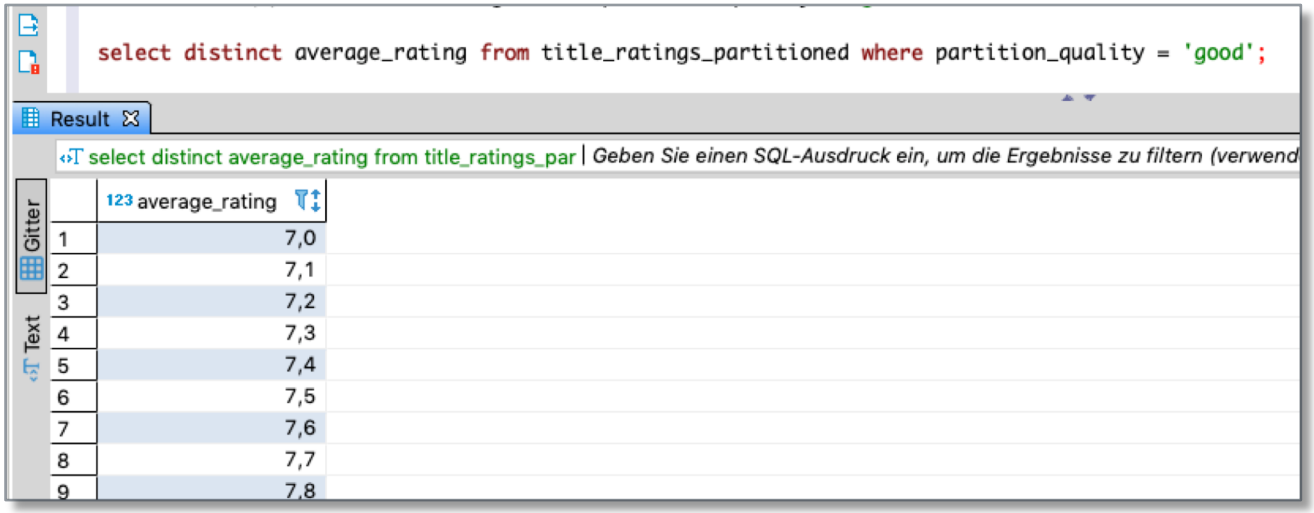
```
INSERT OVERWRITE TABLE title_ratings_partitioned PARTITION(partition_quality='good')
SELECT r.tconst, r.average_rating, r.num_votes FROM title_ratings r WHERE r.average_rating >= 7;

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:



Static Partitioning – INSERT Into Table via Hive

3. Check Success via Hive:



Dynamic Partitioning – **Create Partitioned Table**

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

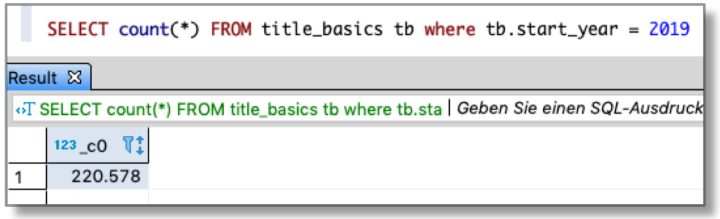
```
CREATE TABLE IF NOT EXISTS title_basics_partitioned (
          tconst STRING,
          title_type STRING,
          primary_title STRING,
          original_title STRING,
          is_adult DECIMAL(1,0),
          start_year DECIMAL(4,0),
          end_year STRING,
          runtime_minutes INT,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

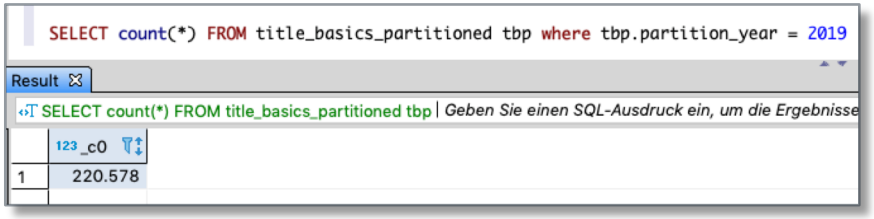

Dynamic Partitioning – INSERT Into Table via Hive

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:





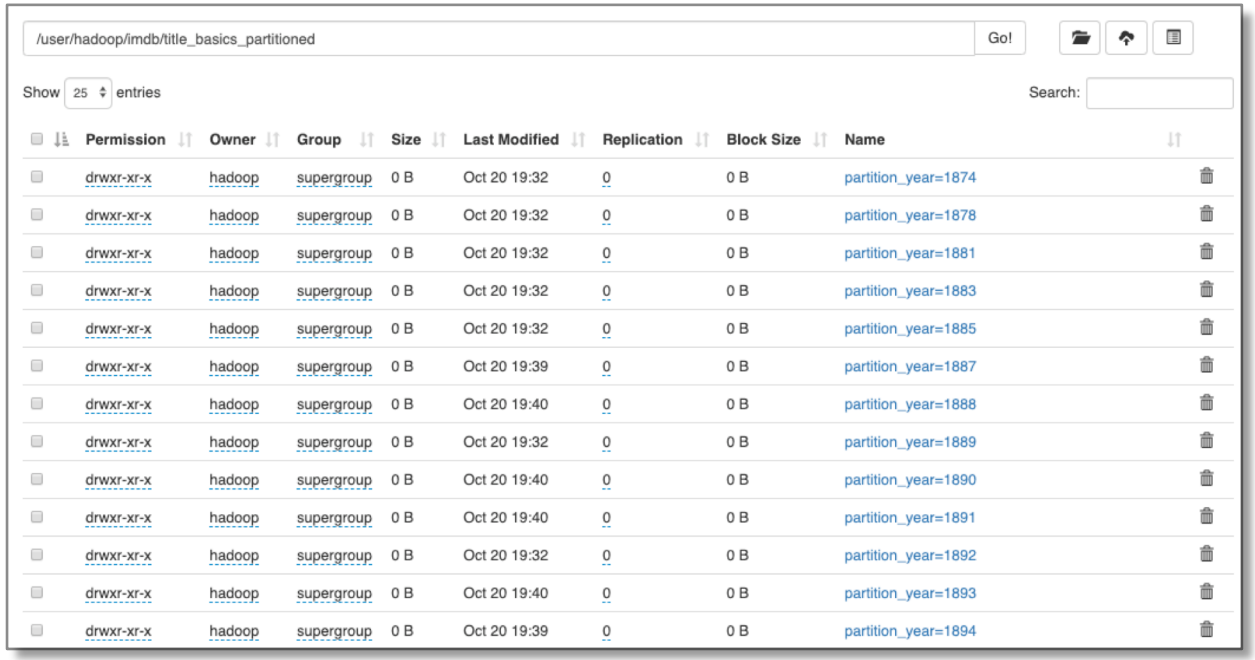
Dynamic Partitioning – **INSERT Into Table via Hive**

3. Check Success on HDFS:

```
hadoop fs -ls /user/hadoop/imdb/title_basics_partitioned
[…]
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:32 /user/hadoop/imdb/title_basics_partitioned/partition_year=1874
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:32 /user/hadoop/imdb/title_basics_partitioned/partition_year=1878
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:32 /user/hadoop/imdb/title_basics_partitioned/partition_year=1881
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:32 /user/hadoop/imdb/title_basics_partitioned/partition_year=1883
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:32 /user/hadoop/imdb/title_basics_partitioned/partition_year=1885
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:39 /user/hadoop/imdb/title_basics_partitioned/partition_year=1887
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:40 /user/hadoop/imdb/title_basics_partitioned/partition_year=1888
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:32 /user/hadoop/imdb/title_basics_partitioned/partition_year=1889
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:40 /user/hadoop/imdb/title_basics_partitioned/partition_year=1890
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:40 /user/hadoop/imdb/title_basics_partitioned/partition_year=1891
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:32 /user/hadoop/imdb/title_basics_partitioned/partition_year=1892
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:40 /user/hadoop/imdb/title_basics_partitioned/partition_year=1893
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:39 /user/hadoop/imdb/title_basics_partitioned/partition_year=1894
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:39 /user/hadoop/imdb/title_basics_partitioned/partition_year=1895
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:39 /user/hadoop/imdb/title_basics_partitioned/partition_year=1896
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:39 /user/hadoop/imdb/title_basics_partitioned/partition_year=1897
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:39 /user/hadoop/imdb/title_basics_partitioned/partition_year=1898
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:39 /user/hadoop/imdb/title_basics_partitioned/partition_year=1899
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:39 /user/hadoop/imdb/title_basics_partitioned/partition_year=1900
drwxr-xr-x   - hadoop supergroup          0 2019-10-20 17:39 /user/hadoop/imdb/title_basics_partitioned/partition_year=1901
[…]
```


Dynamic Partitioning – INSERT Into Table via Hive

4. Check Success via HDFS Web Browser:



Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	hadoop	supergroup	0 B	Oct 20 19:32	0	0 B	partition_year=1874
drwxr-xr-x	hadoop	supergroup	0 B	Oct 20 19:32	0	0 B	partition_year=1878
drwxr-xr-x	hadoop	supergroup	0 B	Oct 20 19:32	0	0 B	partition_year=1881
drwxr-xr-x	hadoop	supergroup	0 B	Oct 20 19:32	0	0 B	partition_year=1883
drwxr-xr-x	hadoop	supergroup	0 B	Oct 20 19:32	0	0 B	partition_year=1885
drwxr-xr-x	hadoop	supergroup	0 B	Oct 20 19:39	0	0 B	partition_year=1887
drwxr-xr-x	hadoop	supergroup	0 B	Oct 20 19:40	0	0 B	partition_year=1888
drwxr-xr-x	hadoop	supergroup	0 B	Oct 20 19:32	0	0 B	partition_year=1889
drwxr-xr-x	hadoop	supergroup	0 B	Oct 20 19:40	0	0 B	partition_year=1890
drwxr-xr-x	hadoop	supergroup	0 B	Oct 20 19:40	0	0 B	partition_year=1891
drwxr-xr-x	hadoop	supergroup	0 B	Oct 20 19:32	0	0 B	partition_year=1892
drwxr-xr-x	hadoop	supergroup	0 B	Oct 20 19:40	0	0 B	partition_year=1893
drwxr-xr-x	hadoop	supergroup	0 B	Oct 20 19:39	0	0 B	partition_year=1894

Exercises IV

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`