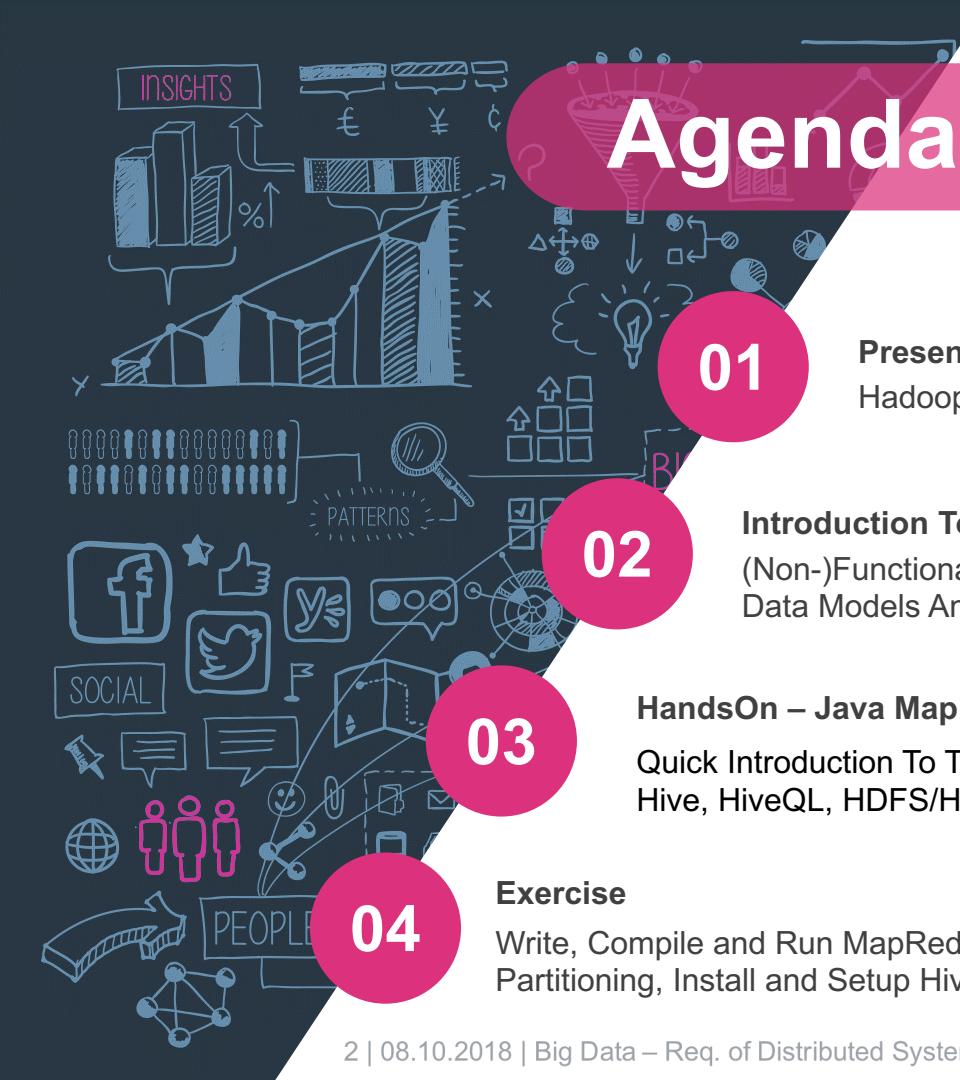


Big Data – Requirements of Distributed Systems, Data Models and Access

Winter Semester 2018,

Cooperative State University Baden-Wuerttemberg



Agenda – 08.10.2018

01

Presentation and Discussion: Exercise Of Last Lecture
Hadoop, HDFS, YARN, MapReduce Example

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, Install and Setup Hive, HDFS/Hive Partitioning, Install and Setup HiveServer2, Use Hive/HiveQL via JDBC



Schedule

| | <i>Lecture Topic</i> | <i>HandsOn</i> |
|--|--|--------------------------------------|
| 01.10.2018 16:00-19:30 Ro. 0.11 | About This Lecture, Introduction to Big Data | Hadoop |
| 08.10.2018 16:00-19:30 Ro. 0.11 | (Non-)Functional Requirements Of Distributed Data-Systems, Data Models and Access | MapReduce, Hive, HiveQL, HiveServer2 |
| 15.10.2018 16:00-19:30 Ro. 0.11 | Challenges Of Distributed Data Systems: Replication | Spark and Jupyter |
| 22.10.2018 16:00-19:30 Ro. 0.11 | Challenges Of Distributed Data Systems: Partitioning | MongoDB |
| 29.10.2018 16:00-19:30 Ro. 0.11 | Batch and Stream Processing | Spark Streaming or Flink |
| 05.11.2018 16:00-19:30 Ro. 0.11 | ETL Workflow And Automation | PDI/Airflow |
| 12.11.2018 16:00-19:30 Ro. 0.11 | Work On Practical Examination | |
| 19.11.2018 16:00-19:30 Ro. 0.11 | Presentation Of Practical Examination | |



Solution – Exercise 01

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/01_hadoop/sample_data/Faust_1.txt  
/user/hadoop/Faust_1.txt
```

2. Use and run default MapReduce Jar (*hadoop-mapreduce-examples-3.1.1.jar*) to calculate **wordcount** for text file „*Faust_1.txt*“.

```
hadoop jar hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.1.jar  
wordcount /user/hadoop/Faust_1.txt /user/hadoop/Faust_1_Output  
[...]  
2018-10-02 10:25:36,818 INFO mapreduce.Job: Running job: job_1538401245375_0003  
2018-10-02 10:25:48,156 INFO mapreduce.Job: map 0% reduce 0%  
2018-10-02 10:25:55,276 INFO mapreduce.Job: map 100% reduce 0%  
2018-10-02 10:26:01,339 INFO mapreduce.Job: map 100% reduce 100%  
2018-10-02 10:26:02,360 INFO mapreduce.Job: Job job_1538401245375_0003 completed successfully  
[...]
```



Solution

Exercise 2:

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

The screenshot shows the Hadoop Resource Manager interface with the title "RUNNING Applications". On the left, there is a sidebar with navigation links for Cluster Metrics, Cluster Nodes Metrics, Scheduler Metrics, and Tools. The main area displays "Cluster Metrics" with the following data:

| Apps Submitted | Apps Pending | Apps Running | Apps Completed | Containers Running | Memory Used | Memory Total | Memory Reserved | VCores Used | VCores Total |
|----------------|--------------|--------------|----------------|--------------------|-------------|--------------|-----------------|-------------|--------------|
| 3 | 0 | 1 | 2 | 2 | 3 GB | 8 GB | 0 B | 2 | 8 |

Below this are sections for "Cluster Nodes Metrics" and "Scheduler Metrics". The "Scheduler Metrics" section shows the following configuration:

| Scheduler Type | Scheduling Resource Type | Minimum Allocation | Maximum Allocation | Maximum Cluster Application |
|--------------------|-------------------------------|-------------------------|-------------------------|-----------------------------|
| Capacity Scheduler | [memory-mb (unit=Mi), vcores] | <memory:1024, vCores:1> | <memory:8192, vCores:4> | 0 |

The "Tools" section contains a table of "Running Applications". One application is listed:

| ID | User | Name | Application Type | Queue | Application Priority | StartTime | FinishTime | State | FinalStatus | Running Containers | Allocated CPU VCores | Allocated Memory MB | Reserved CPU VCores | Reserved Memory MB | % of Queue | % of Cluster | Progress | Tr |
|--------------------------------|--------|------------|------------------|---------|----------------------|--|------------|---------|-------------|--------------------|----------------------|---------------------|---------------------|--------------------|------------|--------------|----------|-----|
| application_1538401245375_0003 | hadoop | word count | MAPREDUCE | default | 0 | Tue Oct 2 10:25:36 +0200 2018 | N/A | RUNNING | UNDEFINED | 2 | 2 | 3072 | 0 | 0 | 37.5 | 37.5 | | App |

At the bottom, it says "Showing 1 to 1 of 1 entries".



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
```

| | |
|---------------------|---|
| <i>Phantasie,</i> | 1 |
| <i>unwanden.</i> | 1 |
| <i>winden,</i> | 1 |
| <i>Offenbarung,</i> | 2 |
| <i>Undene!</i> | 1 |
| <i>Winternächte</i> | 1 |
| <i>derweil</i> | 1 |
| <i>wiederholten</i> | 1 |
| <i>tun</i> | 3 |
| <i>Gestalten.</i> | 1 |

Solution

Exercise 2:

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

The screenshot shows a web browser window for the Hadoop Web Filebrowser at `localhost:9870/explorer.html#/user/hadoop/Faust_1_Output`. The interface includes a navigation bar with links like Hadoop, Overview, Datanodes, Datanode Volume Failures, Snapshot, Startup Progress, and Utilities. Below the bar is a search bar and a table listing directory entries. The table has columns for Permission, Owner, Group, Size, Last Modified, Replication, Block Size, and Name. Two entries are listed: '_SUCCESS' (0 B, Oct 02 10:25, 1 replication, 128 MB block size) and 'part-r-00000' (98.49 KB, Oct 02 10:25, 1 replication, 128 MB block size). A modal window is open over the table, showing the contents of 'part-r-00000'. The modal title is 'part-r-00000' and the path is '~/Downloads'. The modal content area displays the following text:

```
Degen 1
Degen, 1
Dein 8
Deine 2
Deiner 1
Dem 21
Demut 1
Den 51
Den, 2
Denk, 1
Denken. 1
Denkens 1
Denkt 3
```



Solution

Exercise 3:

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

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

2. Use and run default MapReduce Jar (*hadoop-mapreduce-examples-3.1.1.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'  
[...]  
2018-10-02 10:47:39,680 INFO mapreduce.Job: Running job: job_1538401245375_0005  
2018-10-02 10:47:51,905 INFO mapreduce.Job: map 0% reduce 0%  
2018-10-02 10:47:57,966 INFO mapreduce.Job: map 100% reduce 0%  
2018-10-02 10:48:04,023 INFO mapreduce.Job: map 100% reduce 100%  
2018-10-02 10:48:04,031 INFO mapreduce.Job: Job job_1538401245375_0005 completed successfully  
[...]
```



Solution

Exercise 3:

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

The screenshot shows the Hadoop Resource Manager interface with the title "RUNNING Applications". The left sidebar has sections for Cluster Metrics, Cluster Nodes Metrics, Scheduler Metrics, and Tools. The main area displays "Cluster Metrics" with values: Apps Submitted (4), Apps Pending (0), Apps Running (1), Apps Completed (3), Containers Running (2), Memory Used (3 GB), Memory Total (8 GB), Memory Reserved (0 B), VCores Used (2), and VCores Total (8). It also shows "Cluster Nodes Metrics" with Active Nodes (1), Decommissioning Nodes (0), Decommissioned Nodes (0), Lost Nodes (0), Unhealthy Nodes (0), and Rebooted Nodes (0). Under "Scheduler Metrics", it lists the Scheduler Type as Capacity Scheduler, Scheduling Resource Type as [memory-mb (unit=Mi), vcores], Minimum Allocation as <memory:1024, vCores:1>, Maximum Allocation as <memory:8192, vCores:4>, and Maximum Cluster Application as 0. A table titled "Show 20 entries" lists one application: "application_1538401245375_0004" with User "hadoop", Name "grep-search", Application Type "MAPREDUCE", Queue "default", Application Priority "0", Start Time "Tue Oct 2 10:47:14 +0200 2018", Finish Time "N/A", State "RUNNING", Final Status "UNDEFINED", Running Containers "2", Allocated CPU VCores "2", Allocated Memory MB "3072", Reserved CPU VCores "0", Reserved Memory MB "0", % of Queue "37.5", % of Cluster "37.5", and Progress "100%". The bottom status bar says "Showing 1 to 1 of 1 entries".



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**):

The screenshot shows a web browser interface for a Hadoop cluster. The address bar indicates the URL is `localhost:9870/explorer.html#/user/hadoop/Faust_1_Count_Output`. The page title is "Browse Directory". The main content area displays a table of files in the directory `/user/hadoop/Faust_1_Count_Output`. The table has columns for Name, Block Size, Replication, Last Modified, Size, Group, Owner, and Permission. There are two entries:

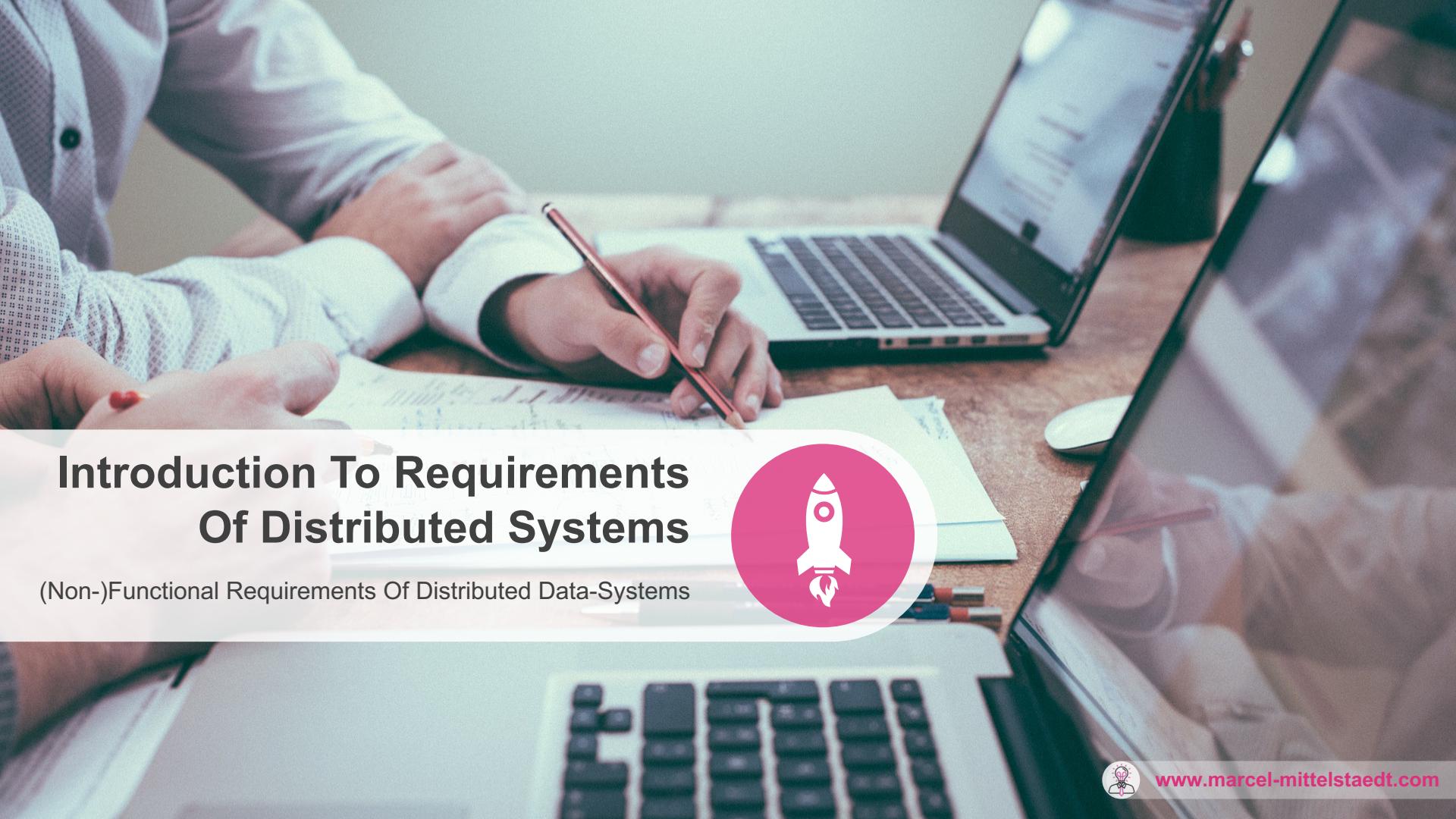
| Name | Block Size | Replication | Last Modified | Size | Group | Owner | Permission |
|--------------|------------|-------------|---------------|------|------------|--------|------------|
| _SUCCESS | 128 MB | 1 | Oct 02 10:48 | 0 B | supergroup | hadoop | -r--r--r-- |
| part-r-00000 | 128 MB | 1 | Oct 02 10:48 | 9 B | supergroup | hadoop | -r--r--r-- |

Below the table, it says "Showing 1 to 2 of 2 entries". At the bottom right of the table, there are "Previous" and "Next" buttons, with "1" highlighted. A modal window is open for the file "part-r-00000". The modal has tabs for "Open" and "Save". The content area shows the text "50 Faust".



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. |



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. *Apache 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?**”*

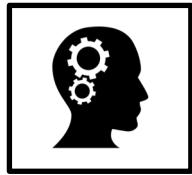
Scalability - Load



*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 between 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 rebalancing 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 system 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:

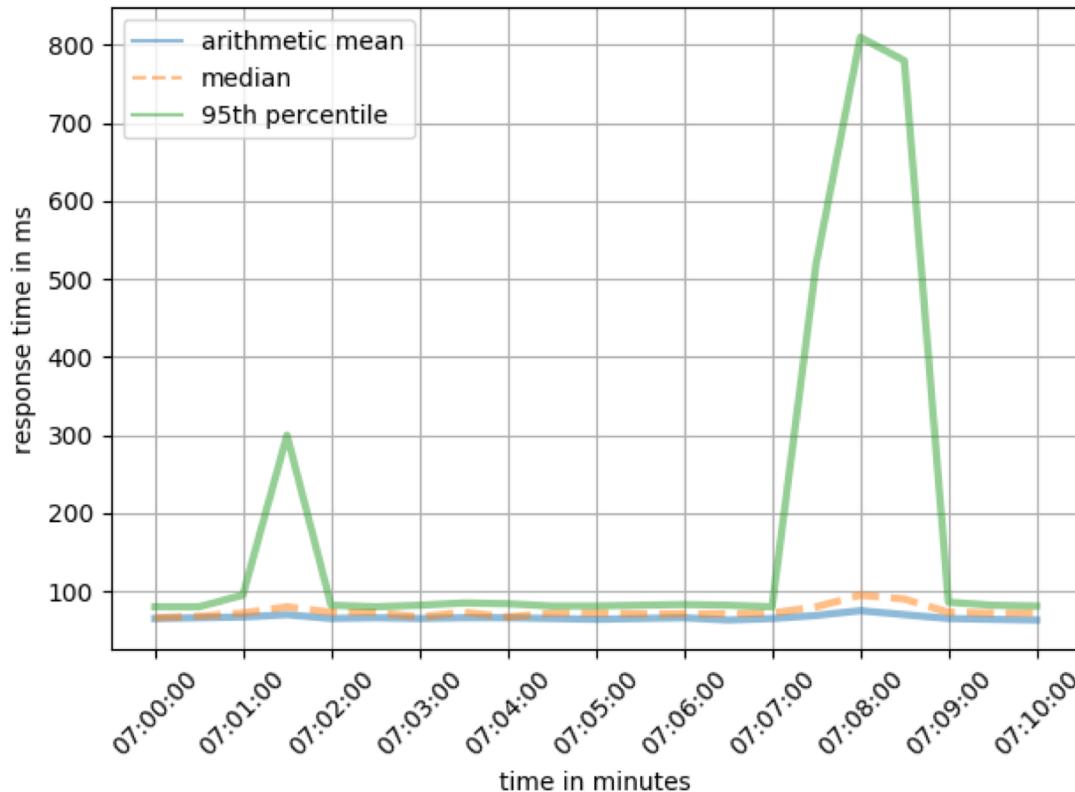
- easy to calculate
- less distorted by outliers

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

- easy to calculate
- not distorted by outliers



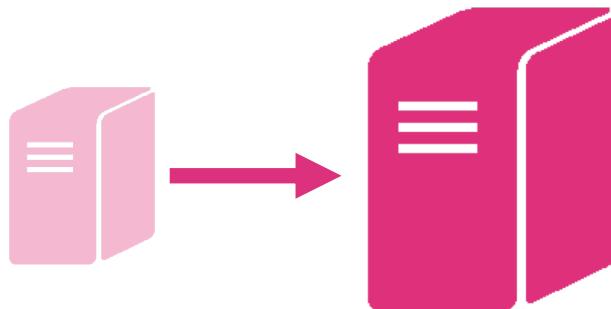
Scalability – Performance Measures



Scalability – Approaches For Scaling

Scale Up/Vertical:

- replace a server by a more powerful one
- easier for a development
- more expensive in terms of hardware



Scale Out/Horizontal:

- distribute the load towards multiple servers instead of one
- development more complex
- cheaper in terms of hardware



→ But it's 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 multiple server was caused)
 - **bottleneck**: a service the whole data-system depends on slows down, 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 automate them
 - **measuring and monitoring** (e.g. performance metrics, error rates) allows to check whether assumptions or constraints are violated at an early 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 understood 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

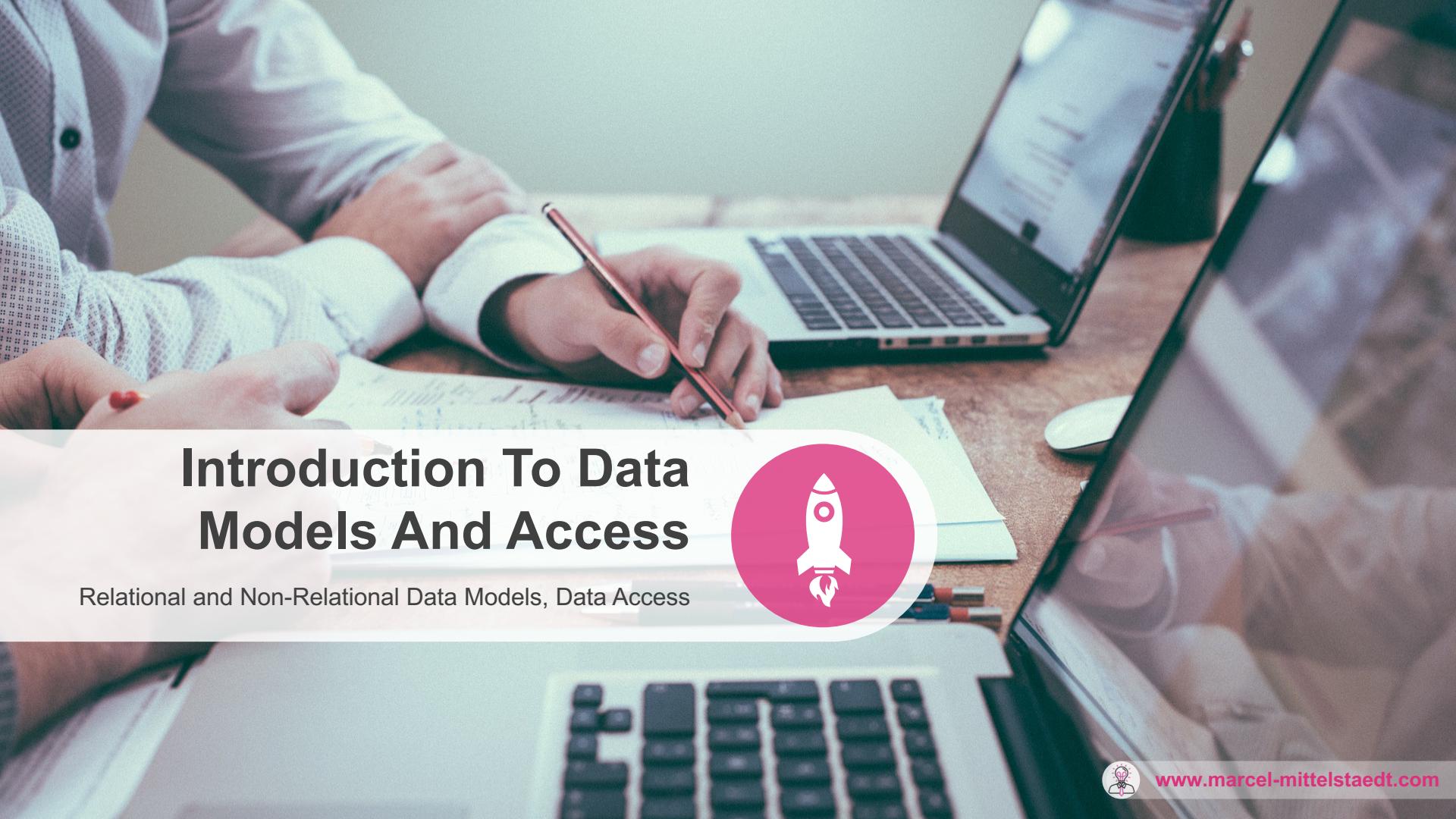
- 2. Simplicity: *make it easy to understand the data-system*

- make use of **abstraction** and **reduce complexity** (less complex doesn't require reducing functionality, 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

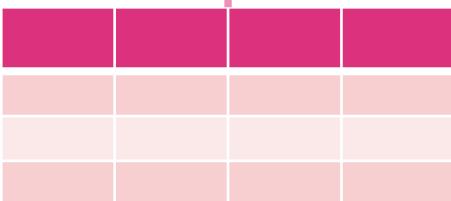
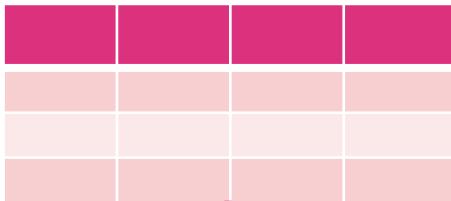
Relational and Non-Relational Data Models, Data Access



Data Models (Relational/Non-Relational)

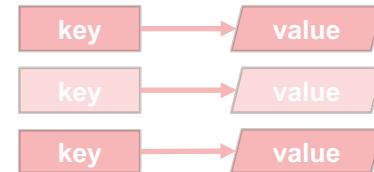
RELATIONAL

Row/Column Based

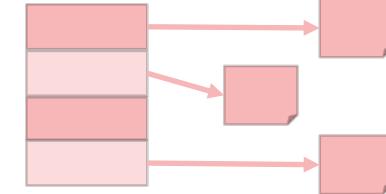


NON-RELATIONAL

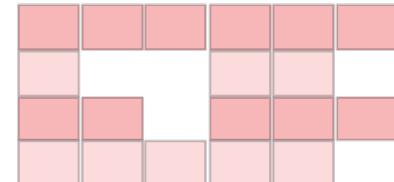
Key-Value



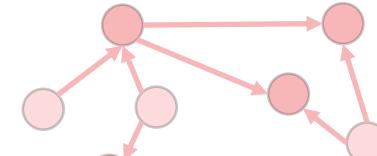
Document



Column Family



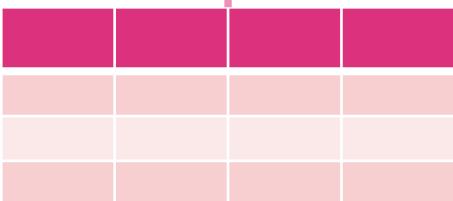
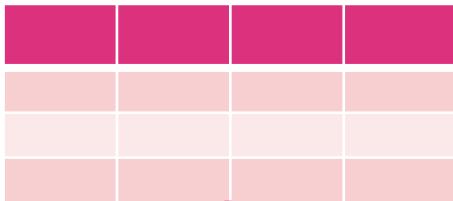
Graph



Relational Data Model

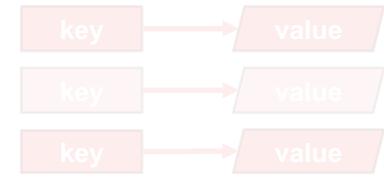
RELATIONAL

Row/Column Based

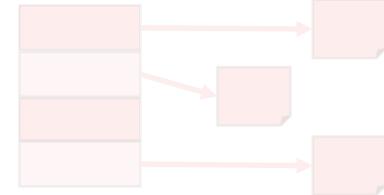


NON-RELATIONAL

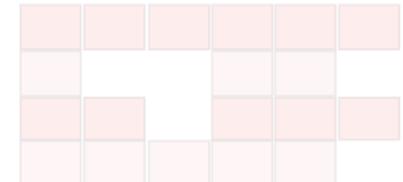
Key-Value



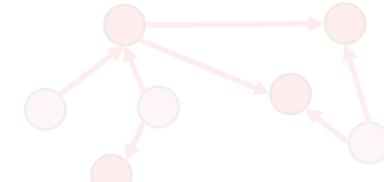
Document



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`)

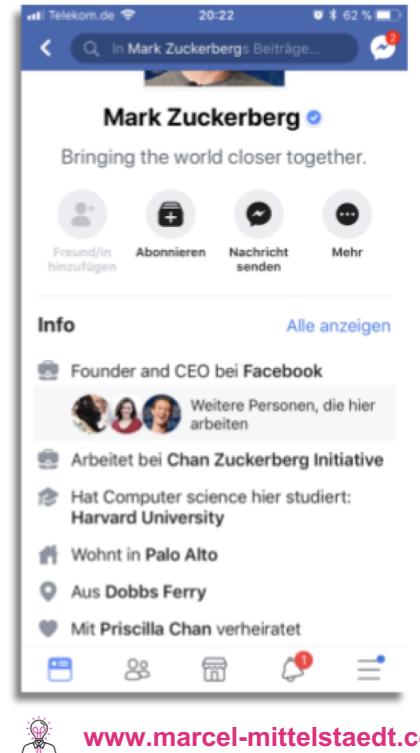
- As people may have:
 - worked in multiple companies
 - studied at multiple universities
 - lived at different cities
- separate tables with *foreign key* (`user_id`):
 - companies
 - universities
 - cities

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

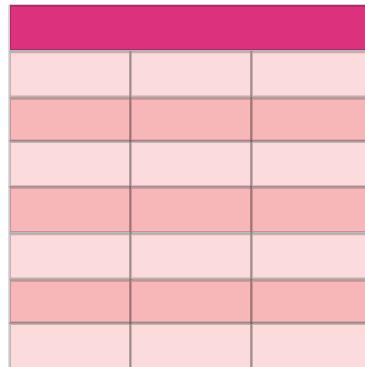
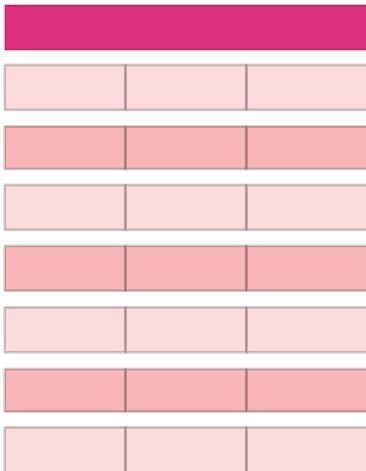
https://en.wikipedia.org/wiki/List_of_relational_database_management_systems



Relational Data Model – Row vs. Column-Based

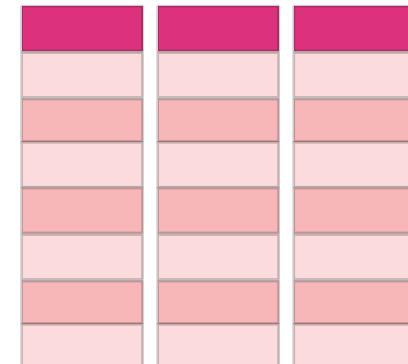
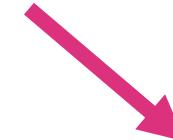
Row-Based:

- rows are stored continuously
- e.g. Oracle, IBM DB2, Microsoft SQL Server, MySQL
- some also support column-based



Column-Based:

- columns are stored continuously
- 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 Codd's** 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:

| | m.tconst | m.original_title | m.start_year | r.average_rating | r.num_votes |
|----|-----------|---|--------------|------------------|-------------|
| 1 | tt0816692 | Interstellar | 2014 | 8,6 | 1.213.141 |
| 2 | tt4154756 | Avengers: Infinity War | 2018 | 8,6 | 492.952 |
| 3 | tt1675434 | Intouchables | 2011 | 8,5 | 635.669 |
| 4 | tt2582802 | Whiplash | 2014 | 8,5 | 571.172 |
| 5 | tt5074352 | Dangal | 2016 | 8,5 | 105.207 |
| 6 | tt1345836 | The Dark Knight Rises | 2012 | 8,4 | 1.331.811 |
| 7 | tt1853728 | Django Unchained | 2012 | 8,4 | 1.153.183 |
| 8 | tt2380307 | Coco | 2017 | 8,4 | 219.957 |
| 9 | tt5311514 | Kimi no na wa. | 2016 | 8,4 | 108.553 |
| 10 | tt2106476 | Jagten | 2012 | 8,3 | 226.819 |
| 11 | tt1832382 | Jodaeiye Nader az Simin | 2011 | 8,3 | 186.217 |
| 12 | tt0993846 | The Wolf of Wall Street | 2013 | 8,2 | 976.551 |
| 13 | tt2096673 | Inside Out | 2015 | 8,2 | 499.721 |
| 14 | tt1291584 | Warrior | 2011 | 8,2 | 389.935 |
| 15 | tt3170832 | Room | 2015 | 8,2 | 288.153 |
| 16 | tt5027774 | Three Billboards Outside Ebbing, Missouri | 2017 | 8,2 | 280.790 |



Relational Data Model - SQL

- **Strengths:**
 - 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

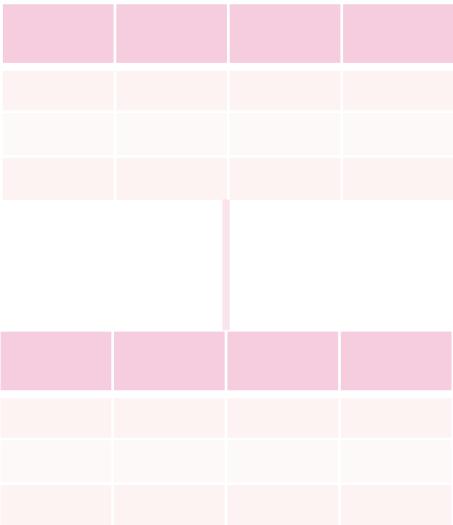
- **Constraints:**
 - **relational data model:**
 - Data Types
 - Primary/Foreign Keys
 - ...
 - highly serves **data integrity**
 - **NoSQL data model:**
 - No or less constraints
 - Freedom but **vulnerable** to *garbage-in – garbage-out*
- **Operations:**
 - **relational data model:** relational algebra
 - mighty language
 - **NoSQL data model:** data-system specific
 - usually less mighty



Non-Relational Data Model

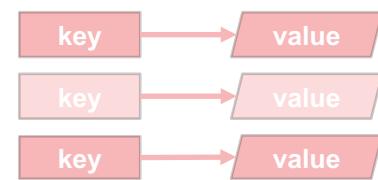
RELATIONAL

Row/Column Based

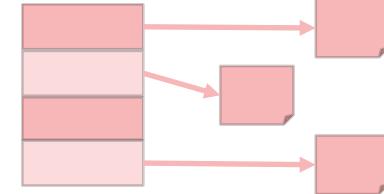


NON-RELATIONAL

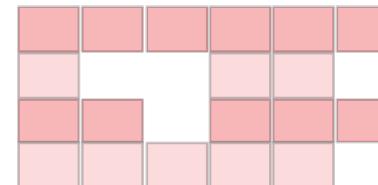
Key-Value



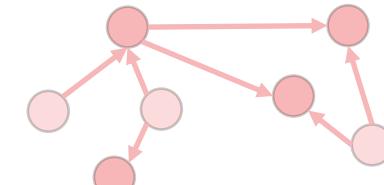
Document



Column Family



Graph

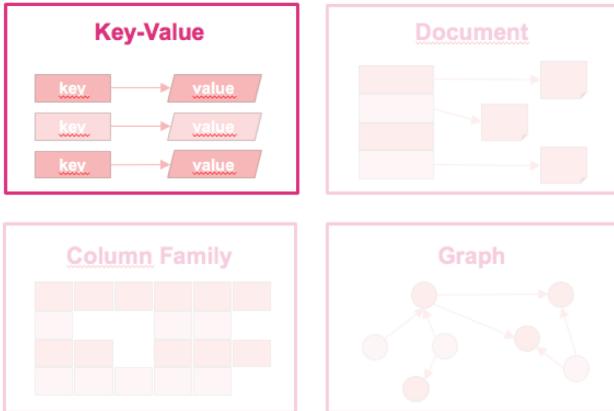


Non-Relational Data Model



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, ...

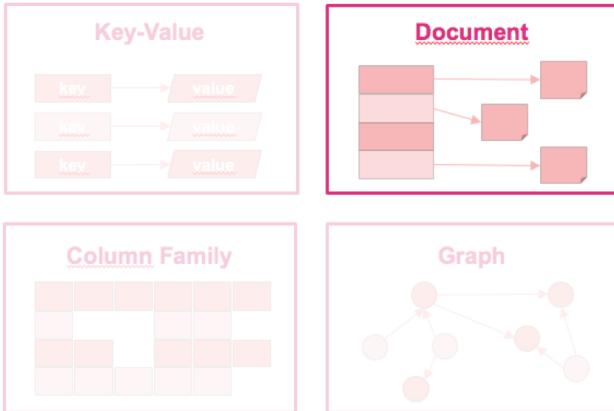
- Strengths:

- **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, ...

- Strengths:

- **schema flexibility** (documents can have different formats)
- **fast inserts and point queries**
- **data locality**
- **easy to replicate and distribute** (e.g. *to achieve load balancing and failure tolerance*)
- **application code simplicity**

- Weaknesses:

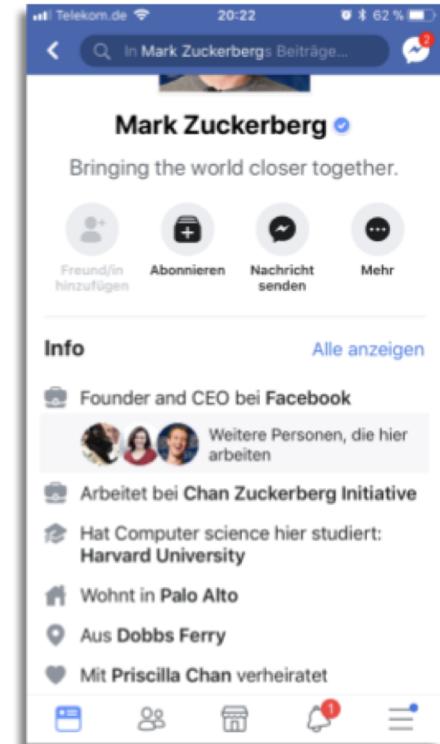
- **updates** on documents are usually **less efficient/IO expensive** (especially if size changes)
- **no or less efficient** and slow:
 - aggregation
 - filtering
 - joining



Non-Relational Data Model – Document

- Example: Facebook Profile Page as document model
- No need to fetch multiple 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

SELECT * FROM user

**SELECT id, first_name, last_name
FROM user WHERE is_verified = true**

**INSERT INTO user(id, first_name,
last_name
VALUES (1, „Mark“, „Zuckerberg“)**

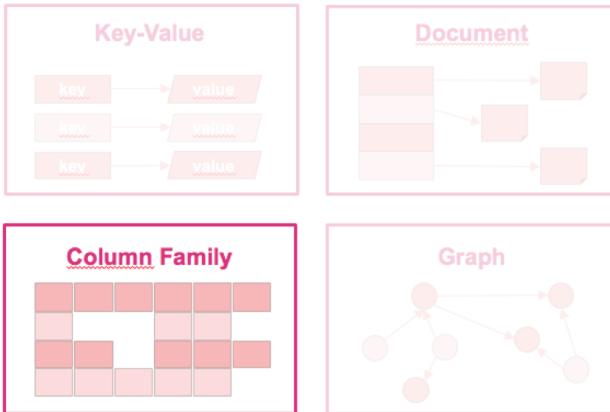
MongoDB

`db.user.find()`

```
db.user.find(  
    { is_verified : true },  
    { id : 1, first_name: 1,  
      last_name: 1 }  
)
```

```
db.user.insertOne(  
    { id : 1,  
      first_name: "Mark",  
      last_name: "Zuckerberg" }  
)
```

Non-Relational Data Model – Column Family



- Examples:

- Cassandra,
- BigTable
- HBase

- Strengths:

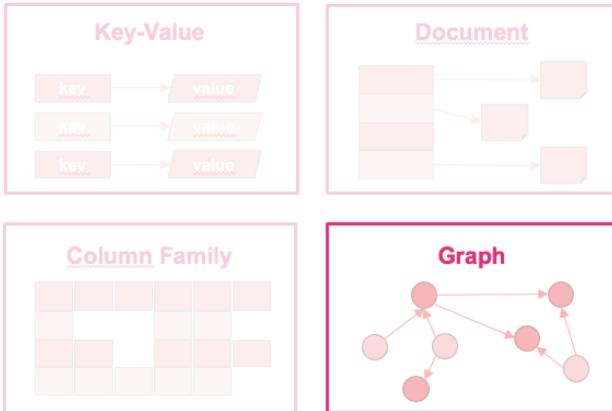
- **schema flexibility** (documents can have different formats)
- **fast inserts and point queries**
- **Fast point queries/value lookups**
- **easy to replicate and distribute** (e.g. *to achieve load balancing and failure tolerance*)

- 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, ...

- Strengths:

- **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 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 parallelized 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 domain, and **returns a list of pairs** in a different domain:

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

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:

$$\text{reduce}(k2, \text{list}(v2)) \rightarrow \text{list}(k2, v3)$$

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



MapReduce – WordCount

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

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

```
1 p1 = [ ("da",1), ("steh",1), ("ich",1), ("nun",1), ("ich",1),
2     ("armer",1), ("tor",1) ];
3 p2 = [ ("und",1), ("bin",1), ("so",1), ("klug",1), ("als",1),
4     ("wie",1), ("zuvor",1) ];
5 ...
6 p6 = [ ("meine",1), ("schüler",1), ("an",1), ("der",1), ("nase",1),
7     ("herum",1)];
```

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



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

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



MapReduce – WordCount

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

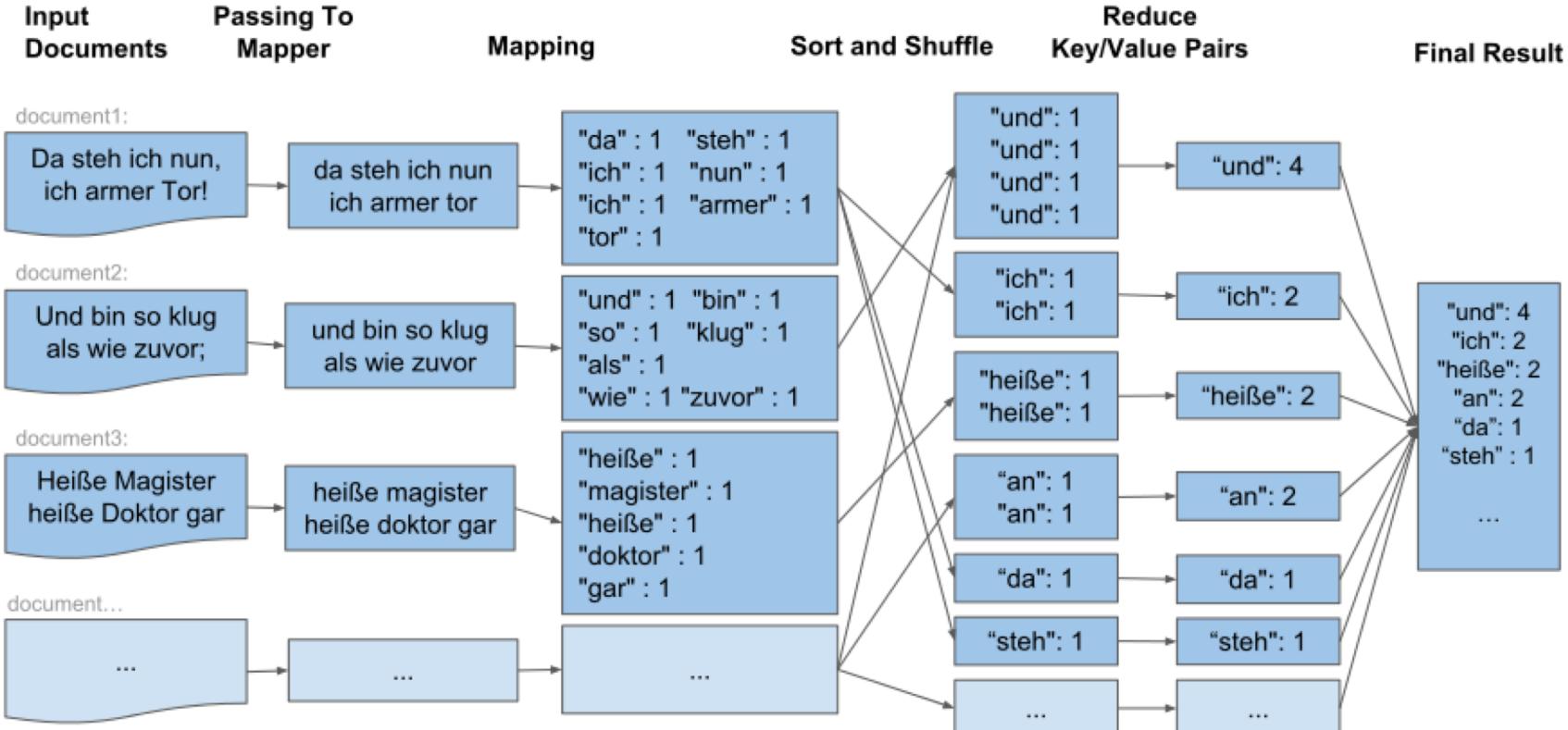
Code Snippet 2.5: MR Example - *Word Count (reduce() Calls)*



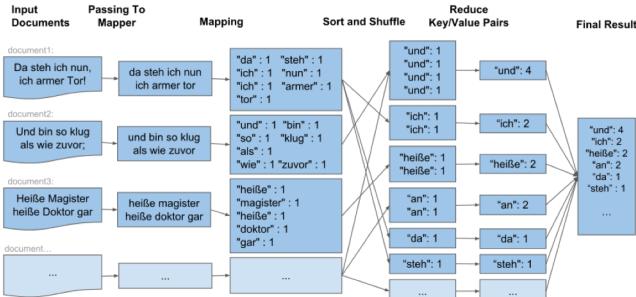
```
1 result = [ ("und", 4), ("ich",2), ("heife",2), ("an", 2), ("da",1),
2   ("steh",1), ("nun",1), ("armer",1), ("tor",1), ("bin",1), ("so",1),
3   ("klug",1), ("als",1), ("wie",1), ("zuvor",1), ("magister",1),
4   ("doktor",1), ("gar",1), ("ziehe",1), ("schon",1), ("die",1),
5   ("zehen",1), ("jahr",1),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

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

MapReduce – Phases



MapReduce – Phases



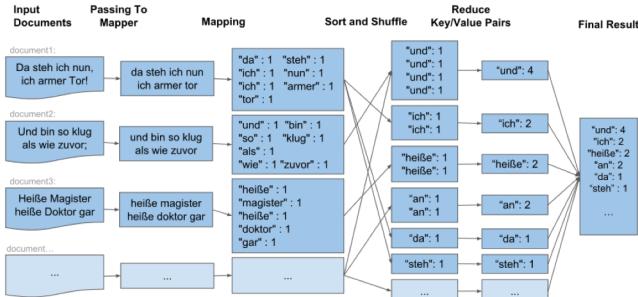
1. Map Phase: Each map function gets multiple key/value pairs and processes each of them separately. All Map processes run independently of each other, facilitating the whole processing to run concurrently on multiple nodes. In our example resulting in several lists of words with an associated count 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, partitioned 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 independently of each other, facilitating the whole processing to run concurrently on multiple nodes. In this way the reduce functions iterate 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 MapReduce 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



www.marcel-mittelstaedt.com

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 MapReduce 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
[...]
```





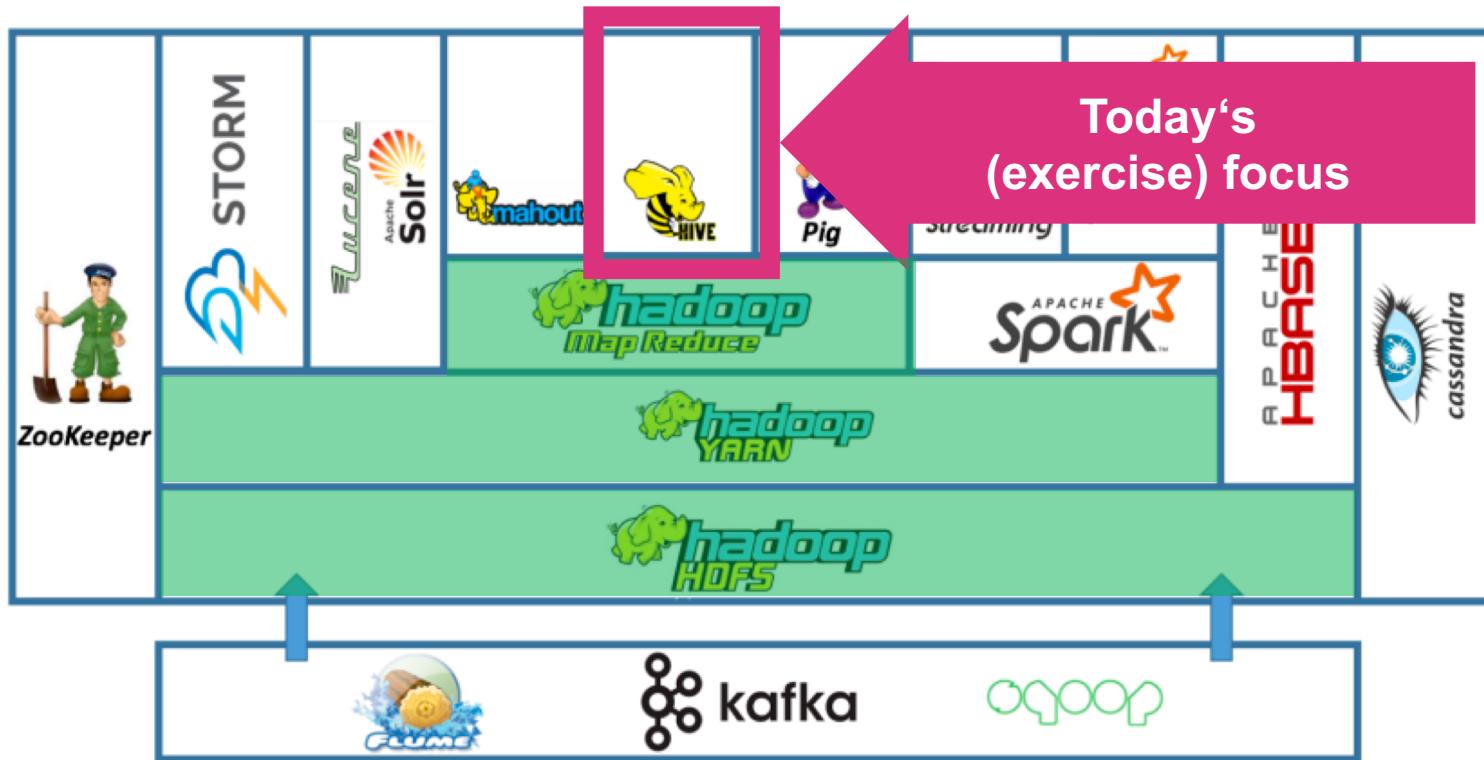
HandsOn – Hive and HiveQL

Quick Introduction To Hive and HiveQL



www.marcel-mittelstaedt.com

The Hadoop Ecosystem



Apache Hive



Apache Hive

- Initial Release in **October 2010**
- written in **Java**
- current Version: **3.1.0**

<http://hive.apache.org>

- **What is Hive:**

- **easy to use** (HiveQL)
- **based on Hadoop** (HDFS, YARN)
- **good scale-out capabilities** (e.g. by partitioning and underlying HDFS and YARN)
- **best used for:**
 - (BigData) **datawarehousing tasks**
 - a **DataLake**
 - **interface** for Analysts, DataScientists, Developers
 - **adhoc (batch) querying, aggregation and analysis** of large amounts of data (**PB!**) and hundreds of nodes

- **What Hive is NOT:**

- **transactional database**
- **highly responsive**



www.marcel-mittelstaedt.com

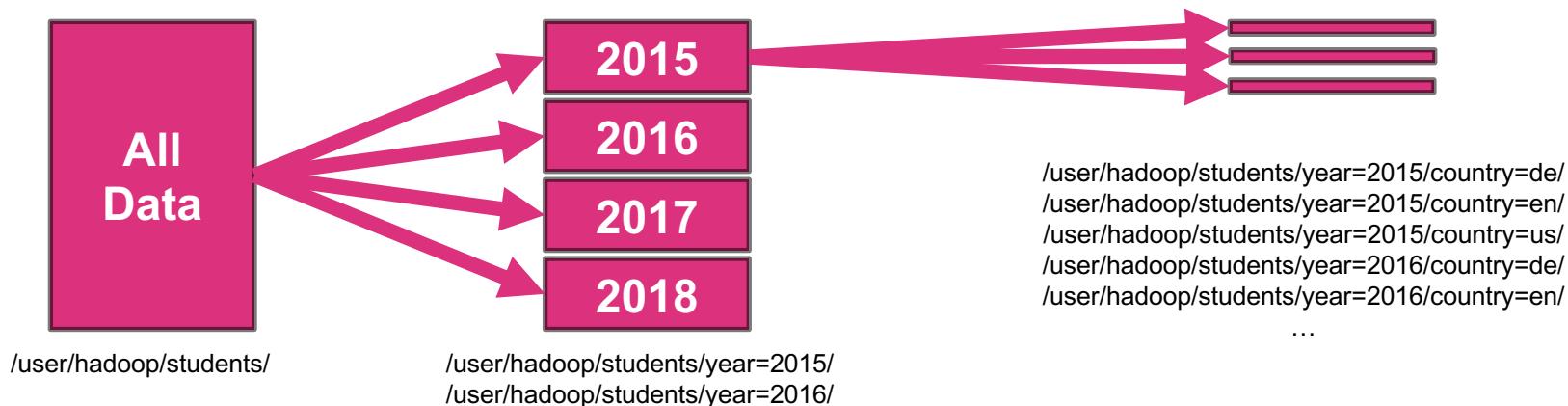
HiveQL

- SQL like **query language**
- Supported by: Hive CLI (*deprecated*), Beeline CLI and most JDBC clients
- Hive supported HDFS file formats:
 - **TextFile** (even compressed by gzip or bzip2)
 - **SequenceFile**
 - **RCFile**
 - **ORC**
 - **Parquet**
 - **Avro**



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**



HDFS/Hive - Wordcount

- Previous WordCount example achieved by using Hive and HiveQL:

```
CREATE TABLE faust (line STRING);

LOAD DATA INPATH '/user/hadoop/faust' OVERWRITE INTO TABLE faust;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
(SELECT explode(split(line, '\\s')) AS word FROM faust) temp
GROUP BY word ORDER BY word;
```

```
SELECT * from word_counts ORDER BY count DESC LIMIT 10;
```



| word | count |
|-------|-------|
| und | 509 |
| die | 463 |
| der | 440 |
| ich | 435 |
| Und | 400 |
| nicht | 346 |
| zu | 319 |
| [...] | |

Exercises Preparation A

Install and Setup Hive



www.marcel-mittelstaedt.com

Install and Setup Hive

1. Get Hive:

```
wget http://ftp.halifax.rwth-aachen.de/apache/hive/hive-3.1.0/apache-hive-3.1  
.0-bin.tar.gz
```

2. Uncompress Hive:

```
tar -xzf apache-hive-3.1.0-bin.tar.gz
```

3. Move Binaries:

```
mv apache-hive-3.1.0-bin /home/hadoop/hive
```



Install and Setup Hive

4. Setup **UNIX** Environment Parameters:

```
vi .bashrc
```



```
export HIVE_HOME=/home/hadoop/hive  
export PATH=$PATH:/home/hadoop/hive/bin
```



```
source .bashrc
```

Install and Setup Hive

5. Verify Hive Installation:

```
hive --version
```



```
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/slf4j/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 3.1.0  
Git git://vgargwork.local/Users/vgarg/repos/hive.apache.master -r bcc7df9582483  
1a8d2f1524e4048dfc23ab98c19  
Compiled by vgarg on Mon Jul 23 16:02:03 PDT 2018  
From source with checksum 147d8070369f8c672407753089777fd
```



Install and Setup Hive

6. Create Hive directories within *HDFS*:

```
hadoop fs -mkdir -p /user/hive/warehouse  
hadoop fs -chmod g+w /user/hive/warehouse
```

```
hadoop fs -mkdir -p /tmp  
hadoop fs -chmod g+w /tmp
```

7. Create Hive Config SH:

```
cp hive/conf/hive-env.sh.template hive/conf/hive-env.sh  
vi hive/conf/hive-env.sh
```



```
export HADOOP_HOME=/home/hadoop/hadoop  
export HIVE_CONF_DIR=/home/hadoop/hive/conf
```

Install and Setup Hive

8. Create Hive Config (*hive-site.xml*):

```
vi hive/conf/hive-site.xml
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?><!--
Licensed to the Apache Software Foundation (ASF) under one or more
contributor license agreements. See the NOTICE file distributed with
this work for additional information regarding copyright ownership.
The ASF licenses this file to You under the Apache License, Version 2.0
(the "License"); you may not use this file except in compliance with
the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
-->
<configuration>
    <property>
        <name>javax.jdo.option.ConnectionURL</name>
        <value>jdbc:derby:;databaseName=/home/hadoop/hive/metastore_db;create=true</value>
        <description>
            JDBC connect string for a JDBC metastore. To use SSL to encrypt/authenticate the connection,
            provide database-specific SSL flag in the connection URL.
            For example, jdbc:postgresql://myhost/db?ssl=true for postgres database.
        </description>
    </property>
```



Install and Setup Hive

```
<property>
    <name>hive.metastore.warehouse.dir</name>
    <value>/user/hive/warehouse</value>
    <description>location of default database for the warehouse
    </description>
</property>
<property>
    <name>hive.metastore.uris</name>
    <value/>
    <description>Thrift URI for the remote metastore. Used by metastore cli
    ent to connect to remote metastore.
    </description>
</property>
<property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>org.apache.derby.jdbc.EmbeddedDriver</value>
    <description>Driver class name for a JDBC metastore</description>
</property>
<property>
    <name>javax.jdo.PersistenceManagerFactoryClass</name>
    <value>org.datanucleus.api.jdo.JDOPersistenceManagerFactory</value>
    <description>class implementing the jdo persistence</description>
</property>
</configuration>
```



Install and Setup Hive

9. Edit YARN MapRed Config (*mapred-site.xml*) for higher memory usage:

```
vi hadoop/etc/hadoop/mapred-site.xml
```

```
<property>
    <name>mapreduce.map.memory.mb</name>
    <value>3072</value>
</property>
<property>
    <name>mapreduce.reduce.memory.mb</name>
    <value>3072</value>
</property>
```

10. Restart DFS and YARN:

```
stop-all.sh
start-dfs.sh
Start-yarn.sh
```



Install and Setup Hive

11. By default Hive makes use of *Derby* as a metastore database, so we need to initialize *Derby*:

```
hive/bin/schematool -initSchema -dbType derby
```



```
[...]
Starting metastore schema initialization to 3.1.0
Initialization script hive-schema-3.1.0.derby.sql
[...]
Initialization script completed
schemaTool completed
```

Install and Setup Hive

12. Test if Hive Installation and Configuration is successful. Start Hive:

```
hive
```



```
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/slf4j/  
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 = c120d0b1-9025-43db-96e4-48ccfb875f1a  
  
Logging initialized using configuration in jar:file:/home/hadoop/hive/lib/hive-common-3.1.0.jar  
!/hive-log4j2.properties Async: true  
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider us  
ing a different execution engine (i.e. spark, tez) or using Hive 1.X releases.  
Hive Session ID = fdd6f06a-d4e4-48e6-8971-997e8a0a8e2c  
hive>
```



Install and Setup Hive

13. Execute First SQL Query:

```
hive> show databases;  
OK  
default  
Time taken: 0.083 seconds, Fetched: 1 row(s)  
hive>
```





Hive: Create and Work with External Tables

Using public dataset of IMDb.com



www.marcel-mittelstaedt.com

Get IMDb Data And Move It To HDFS

1. Get **IMDb Data** (<https://www.imdb.com/interfaces/>):

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

2. Uncompress IMDb Data:

```
gunzip title.basics.tsv.gz
gunzip title.ratings.tsv.gz
```

3. Create HDFS Directories for IMDb Data:

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



Create External Tables In Hive

4. Transfer IMDb data files to HDFS:

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

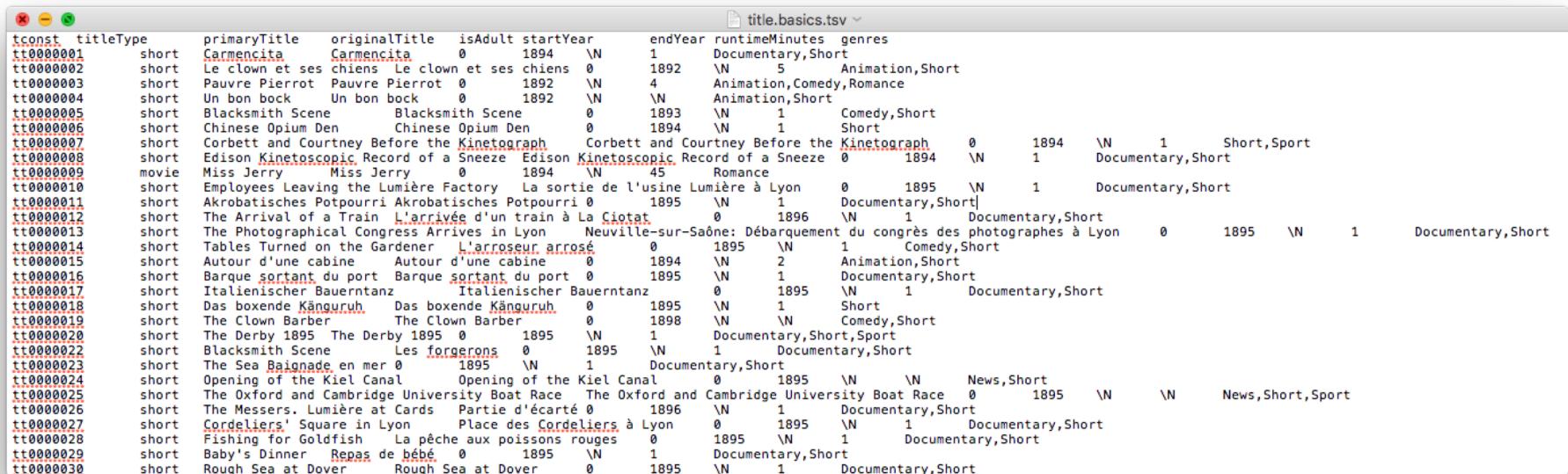
5. Create External Table **imdb_ratings** (file *title.ratings.tsv*) in Hive:

```
hive > CREATE EXTERNAL TABLE IF NOT EXISTS imdb_ratings(  
        tconst STRING,  
        average_rating DECIMAL(2,1),  
        num_votes BIGINT  
) COMMENT 'IMDb Ratings'  
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS  
TEXTFILE LOCATION '/user/hadoop/imdb/ratings';
```

| tconst | averageRating | numVotes |
|-----------|---------------|----------|
| tt0000001 | 5.8 | 1416 |
| tt0000002 | 6.4 | 167 |
| tt0000003 | 6.6 | 1013 |
| tt0000004 | 6.4 | 100 |
| tt0000005 | 6.2 | 1712 |
| tt0000006 | 5.6 | 87 |
| tt0000007 | 5.5 | 571 |
| tt0000008 | 5.6 | 1520 |
| tt0000009 | 5.6 | 68 |
| tt0000010 | 6.9 | 5075 |
| tt0000011 | 5.4 | 208 |
| tt0000012 | 7.4 | 8479 |
| tt0000013 | 5.7 | 1297 |
| tt0000014 | 7.2 | 3683 |
| tt0000015 | 6.2 | 642 |

Create External Tables In Hive

6. Create External Table `imdb_movies` for file `title.basics.tsv` in Hive:



| tconst | titleType | primaryTitle | originalTitle | isAdult | startYear | endYear | runtimeMinutes | genres |
|-----------|-----------|---|---|---------|-----------|---------|----------------|--------------------------|
| tt0000001 | short | Carmencita | Carmencita | 0 | 1894 | \N | 1 | Documentary,Short |
| tt0000002 | short | Le clown et ses chiens | Le clown et ses chiens | 0 | 1892 | \N | 5 | Animation,Short |
| tt0000003 | short | Pauvre Pierrot | Pauvre Pierrot | 0 | 1892 | \N | 4 | Animation,Comedy,Romance |
| tt0000004 | short | Un bon bock | Un bon bock | 0 | 1892 | \N | 1 | Animation,Short |
| tt0000005 | short | Blacksmith Scene | Blacksmith Scene | 0 | 1893 | \N | 1 | Comedy,Short |
| tt0000006 | short | Chinese Opium Den | Chinese Opium Den | 0 | 1894 | \N | 1 | Short |
| tt0000007 | short | Corbett and Courtney Before the Kinetograph | Corbett and Courtney Before the Kinetograph | 0 | 1894 | \N | 1 | Short,Sport |
| tt0000008 | short | Edison Kinetoscopic Record of a Sneeze | Edison Kinetoscopic Record of a Sneeze | 0 | 1894 | \N | 1 | Documentary,Short |
| tt0000009 | movie | Miss Jerry | Miss Jerry | 0 | 1894 | \N | 45 | Romance |
| tt0000010 | short | Employees Leaving the Lumière Factory | La sortie de l'usine Lumière à Lyon | 0 | 1895 | \N | 1 | Documentary,Short |
| tt0000011 | short | Akrobatisches Potpourri | Akrobatisches Potpourri | 0 | 1895 | \N | 1 | Documentary,Short |
| tt0000012 | short | The Arrival of a Train | L'arrivée d'un train à La Ciotat | 0 | 1896 | \N | 1 | Documentary,Short |
| tt0000013 | short | The Photographical Congress Arrives in Lyon | Neuville-sur-Saône: Débarquement du congrès des photographes à Lyon | 0 | 1895 | \N | 1 | Documentary,Short |
| tt0000014 | short | Tables Turned on the Gardener | L'arroseur arrosé | 0 | 1895 | \N | 1 | Comedy,Short |
| tt0000015 | short | Autour d'une cabine | Autour d'une cabine | 0 | 1894 | \N | 2 | Animation,Short |
| tt0000016 | short | Barque sortant du port | Barque sortant du port | 0 | 1895 | \N | 1 | Documentary,Short |
| tt0000017 | short | Italienischer Bauerntanz | Italienischer Bauerntanz | 0 | 1895 | \N | 1 | Documentary,Short |
| tt0000018 | short | Das boxende Känguru | Das boxende Känguru | 0 | 1895 | \N | 1 | Short |
| tt0000019 | short | The Clown Barber | The Clown Barber | 0 | 1898 | \N | 1 | Comedy,Short |
| tt0000020 | short | The Derby 1895 | The Derby 1895 | 0 | 1895 | \N | 1 | Documentary,Short,Sport |
| tt0000022 | short | Blacksmith Scene | Les forgerons | 0 | 1895 | \N | 1 | Documentary,Short |
| tt0000023 | short | The Sea Baignade en mer | \N | 0 | 1895 | \N | 1 | Documentary,Short |
| tt0000024 | short | Opening of the Kiel Canal | Opening of the Kiel Canal | 0 | 1895 | \N | 1 | News,Short |
| tt0000025 | short | The Oxford and Cambridge University Boat Race | The Oxford and Cambridge University Boat Race | 0 | 1895 | \N | 1 | News,Short,Sport |
| tt0000026 | short | The Messers. Lumière at Cards | Partie d'écarté | 0 | 1896 | \N | 1 | Documentary,Short |
| tt0000027 | short | Cordeliers' Square in Lyon | Place des Cordeliers à Lyon | 0 | 1895 | \N | 1 | Documentary,Short |
| tt0000028 | short | Fishing for Goldfish | La pêche aux poissons rouges | 0 | 1895 | \N | 1 | Documentary,Short |
| tt0000029 | short | Baby's Dinner | Repas de bébé | 0 | 1895 | \N | 1 | Documentary,Short |
| tt0000030 | short | Rough Sea at Dover | Rough Sea at Dover | 0 | 1895 | \N | 1 | Documentary,Short |



Create External Tables In Hive

6. Create External Table `imdb_movies` for file `title.basics.tsv` in Hive:

```
hive > CREATE EXTERNAL TABLE IF NOT EXISTS imdb_movies(
        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
    ) COMMENT 'IMDb Movies' ROW FORMAT DELIMITED FIELDS TERMINATED
    BY '\t' STORED AS TEXTFILE LOCATION '/user/hadoop/imdb/name';
```





Exercises I

Hive: Create and Work with External Tables



HDFS and HiveQL Exercises - IMDB

1. Execute Tasks of previous HandsOn Slides
2. Download <https://datasets.imdbws.com/name.basics.tsv.gz>
3. Create HDFS Directory **/user/hadoop/imdb/actors/names.tsv** for file name.basics.tsv
4. Create External Hive Table **imdb_actors** for name.tsv
5. Use HiveQL to answer following questions:
 - a) How many movies are within the IMDB dataset?
 - b) Who is the oldest actor/writer/... within the dataset?
 - 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 2000
 - have an average rating better than 8
 - have been voted more than 100.000 times
 - d) How many movies are in list of c)?



HDFS and HiveQL Exercises - IMDB

5. Use HiveQL to answer following questions:

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.

