

Решение – Упражнение HiveQL,  
создание и работа с внешними  
таблицами на данных IMDb

# HDFS и Hive QL упражнения - IMDB

1. Выполните все действия на слайдах выше
2. Скачать <https://datasets.imdbws.com/name.basics.tsv.gz>
3. Создать каталог HDFS `/user/hadoop/imdb/name_basics/` для файла **name.basics.tsv**
4. Создайте внешнюю Hive таблицу `name_basics` для **name.basics.tsv**
5. Используйте HiveQL, чтобы ответить на следующие вопросы:
  - a) Сколько фильмов и сериалов находится в наборе данных IMDB?
  - b) Кто самый молодой актер/сценарист/... в наборе данных?
  - c) Создайте список (`tconst`, `original_title`, `start_year`, `average_rating`, `num_votes`), который состоит из:
    - фильм вышел в 2010 году или позднее;
    - фильм имеет средний рейтинг, равный или превышающий 8,1
    - проголосовали более 100 000 раз
  - d) Сколько фильмов находится в списке c)?

# HDFS и Hive QL упражнения - IMDB

5. Используйте **HiveQL**, чтобы ответить на следующие вопросы::

е) Мы хотим знать, какие годы были великими для кинематографа.

Создайте список с одной строкой в год и соответствующим количеством фильмов, которые:

- имеют средний рейтинг выше 8;

- были проголосованы более 100 000 раз в порядке убывания количества фильмов.

# Решение индивидуального задания



## Предварительные требования:

- Запустить экземпляр VM
- 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`

# Решение индивидуального задания



## Задание 1-4:

### 1. Скачиваем и распаковываем данные <https://datasets.imdbws.com/name.basics.tsv.gz>

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

### 2. Создаем HDFS каталог /user/hadoop/imdb/name\_basics/ для файла name.basics.tsv

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

### 3. Перемещаем TSV-файл в HDFS каталог

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

# Решение индивидуального задания

## Задание 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' STORED  
    AS TEXTFILE LOCATION '/user/hadoop/imdb/name_basics'  
TBLPROPERTIES ('skip.header.line.count'='1');
```

# Решение индивидуального задания

## Задание 5:

a) Сколько фильмов и сколько сериалов содержится в наборе данных IMDB?

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

b) Кто самый молодой актер/сценарист/... в наборе данных?

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

# Решение индивидуального задания

## Задание 5:

a) Сколько фильмов и сколько сериалов содержится в наборе данных IMDB?

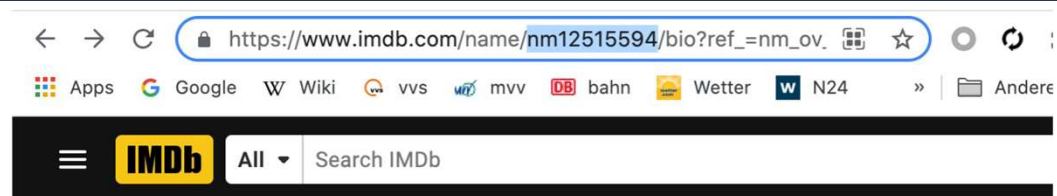
```
hive > SELECT m.title_type, count(*)  
FROM title_basics m GROUP BY m.title_type;
```

b) Кто самый молодой актер/сценарист/... в наборе данных?

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



# Решение индивидуального задания



Noah Lira

Edit

## Biography

Showing all 6 items

Jump to: [Overview \(2\)](#) | [Mini Bio \(1\)](#) | [Family \(2\)](#) | [Trivia \(1\)](#)

### Overview (2)

**Born** [February 14, 2021](#) in [São Paulo, São Paulo, Brazil](#)

**Nickname** [Noah](#)

### Mini Bio (1)

Noah Lira was born on February 14, 2021 in São Paulo, São Paulo, Brazil. He is an actor and cinematographer, known for [Invento na Hora](#) (2021).

### Family (2)

**Parents** [Lucas Lira](#)  
[Sunaika Bruna](#)

**Relatives** [Kiara Lira](#) (sibling)

### Trivia (1)

Noah is son Lucas Lira and Sunaika Bruna.

See also

# Решение индивидуального задания

## Задание 5:

С) Создать список (*m.tconst*, *m.original\_title*, *m.start\_year*, *r.average\_rating*, *r.num\_votes*) фильмов, включающий следующие элементы:

- фильм вышел в 2010 году или позднее.
- фильм имеет средний рейтинг равный или выше 8,1.
- проголосовали за фильм более 100 000 раз.

```
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;
```

# Решение индивидуального задания

## Задание 5:

d) Сколько фильмов в списке 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;
```

# Решение индивидуального задания

## Задание 5:

d) Мы хотим знать, какие годы были великими для кино.

Создайте список с одной строкой в год и соответствующим количеством фильмов, которые:

- имеют средний рейтинг выше 8
- проголосовали за фильм более 100 000 раз в порядке убывания по количеству фильмов.

```
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;
```

# Решение индивидуального задания

## Задание 5:

d) Итак, 1995 год кажется действительно хорошим годом для кино, было выпущено 8 действительно хороших фильмов, но какие?

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

# Введение в модели данных и доступ

# Реляционная модель данных

Originally introduced by Edgar Frank Codd in 1970



Разработчики/пользователи могут легко указать, какую информацию содержит база данных и какую информацию они хотят получить.

База данных позаботится об описании, хранении и извлечении данных

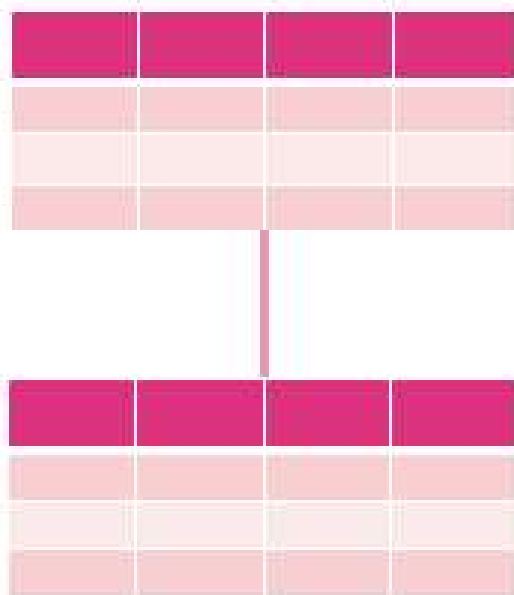
**Идея:** скрыть детали реализации (представление данных в хранилище данных)

**Путем предоставления:** декларативного и читаемого на схеме интерфейса

# Модели данных (реляционные/нереляционные)

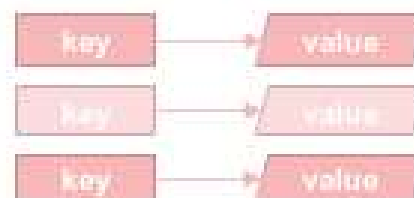
## RELATIONAL

### Row/Column Based

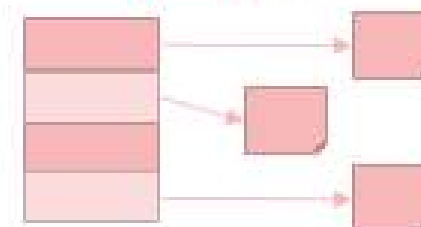


## NON-RELATIONAL

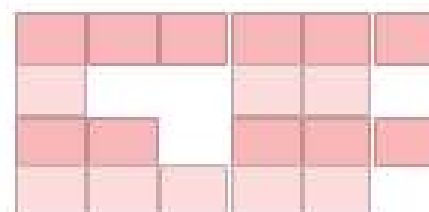
### Key-Value



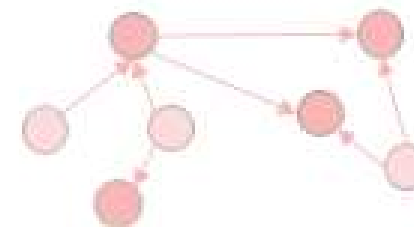
### Document



### Column Family



### Graph



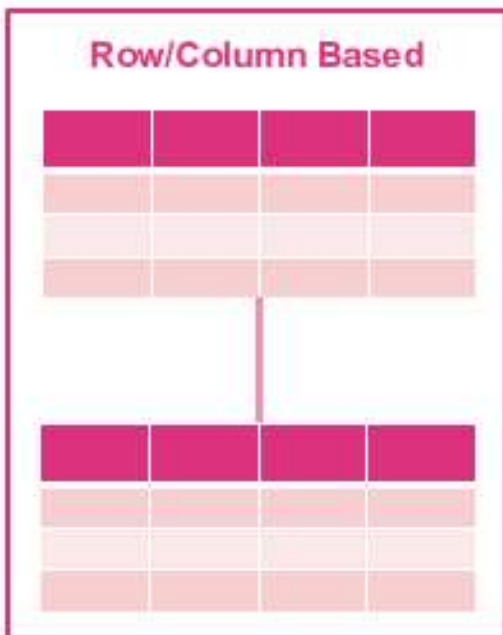


# Реляционная модель данных

Originally introduced by Edgar Frank Codd in 1970

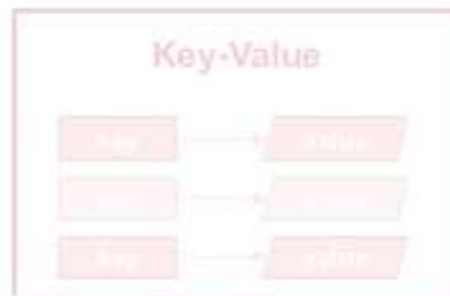
## RELATIONAL

### Row/Column Based

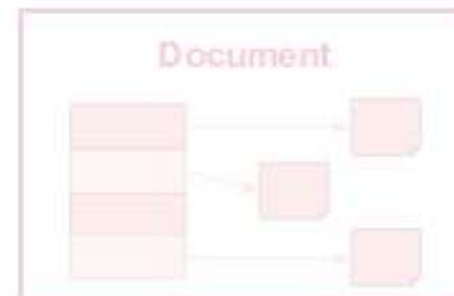


## NON-RELATIONAL

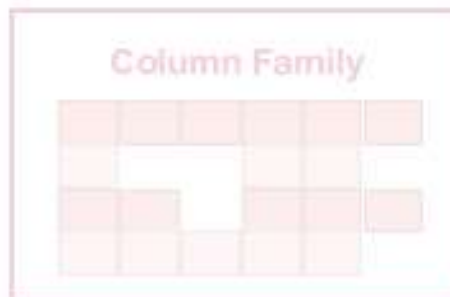
### Key-Value



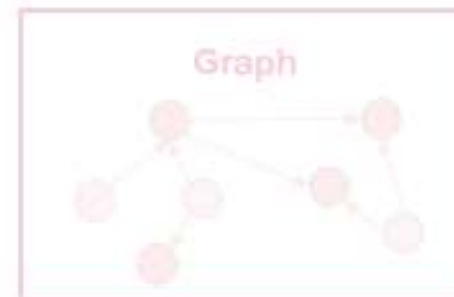
### Document



### Column Family



### Graph



**Идея:** скрыть детали реализации (представление данных в хранилище данных)

**Путем предоставления:** декларативного и читаемого на схеме интерфейса

# Реляционная модель данных

Originally introduced by Edgar Frank Codd in 1970



Разработчики/пользователи могут легко указать, какую информацию содержит база данных и какую информацию они хотят получить.

База данных позаботится об описании, хранении и извлечении данных

**Идея:** скрыть детали реализации (представление данных в хранилище данных)

**Путем предоставления:** декларативного и читаемого на схеме интерфейса

# Реляционная модель данных — список программного обеспечения

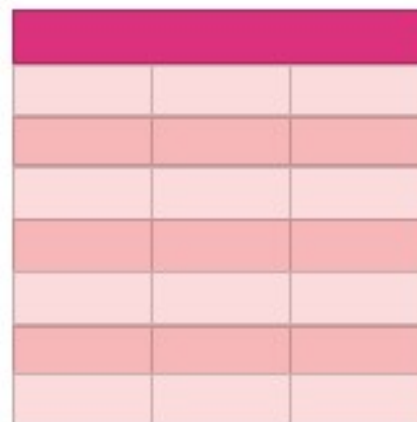
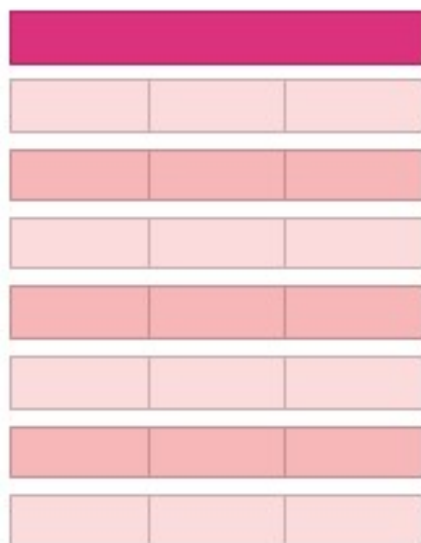
## List of Software [\[ edit \]](#)

- |                               |                             |   |  |   |  |
|-------------------------------|-----------------------------|---|--|---|--|
| • 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<br>(formerly known as Sybase<br>Advantage Database Server) |
| • Aster Data                  | • EnterpriseDB              | • InterBase   | • MySQL                                      | • Postgres Plus Advanced Server   | • Teradata   |
| • Amazon Aurora               | • eXtremeDB                 | • InterSystems Caché  | • Netezza                                    | • Progress Software   | • Tiberio  |
| • Altibase                    | • FileMaker Pro             | • LibreOffice Base  | • NexusDB                                    | • RDM Embedded  | • TimesTen   |
| • CA Datacom                  | • Firebird                  | • Linter  | • NonStop SQL                                | • RDM Server  | • Trafodion  |
| • CA IDMS                     | • FrontBase                 | • MariaDB   | • NuoDB                                      | • R:Base  | • txtSQL   |
| • Clarion                     | • Google Fusion Tables      | • MaxDB   | • Omnis Studio                               | • SAND CDBMS  | • Unisys RDMS 2200   |
| • ClickHouse                  | • Greenplum                 | • MemSQL  | • Openbase                                   | • SAP HANA  | • UniData  |
| • Clustrix                    | • GroveSite                 | • Microsoft Access  | • OpenLink Virtuoso (Open Source<br>Edition) | • SAP Adaptive Server Enterprise  | • UniVerse   |
| • CSQL                        | • H2                        | • Microsoft Jet Database Engine<br>(part of Microsoft Access) | • OpenLink Virtuoso Universal<br>Server      | • SAP IQ (formerly known as<br>Sybase IQ)   | • Vectorwise   |
| • CUBRID                      | • Helix database            | • Microsoft SQL Server  | • OpenOffice.org Base                        | • SQL Anywhere (formerly known<br>as Sybase Adaptive Server<br>Anywhere and Watcom SQL) | • Vertica  |
| • DataEase                    | • HSQLDB                    | • Microsoft SQL Server Express                                | • Oracle                                     | • solidDB   | • VoltDB   |
| • Database Management Library | • IBM DB2                   | • SQL Azure (Cloud SQL Server)                                | • Oracle Rdb for OpenVMS                     |   |  |
| • Dataphor                    | • IBM Lotus Approach        |   |  |   |  |

[https://en.wikipedia.org/wiki/List\\_of\\_relational\\_database\\_management\\_systems](https://en.wikipedia.org/wiki/List_of_relational_database_management_systems)

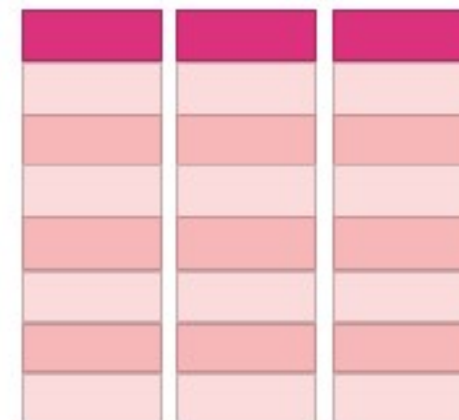
### Row-Based:

- строки хранятся непрерывно
- Oracle, IBM DB2, Microsoft SQL Server, MySQL



### Column-Based:

- столбцы хранятся непрерывно
- Hbase, Parquet, SAP HANA, Teradata



## Реляционная модель данных — на основе строк(Row) и столбцов(Column-Based)

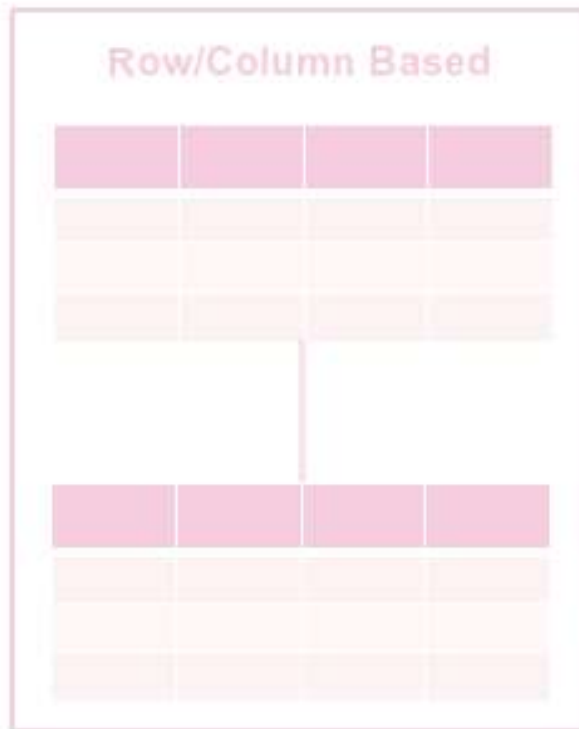
Operation	Row-Based	Column-Based
Compression	Low	High
Column Scans	Slow (Multiple Reads)	Fast (One Read)
Insert Of Records	Fast (One Insert)	Slow (Multiple Inserts)
Single Record Queries	Fast (One Read)	Slow (Multiple Reads)
Single Column Aggregation	Slow (Full Table Scan)	Fast (Only Column Scan)
Typical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

- **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

# Нереляционная модель данных

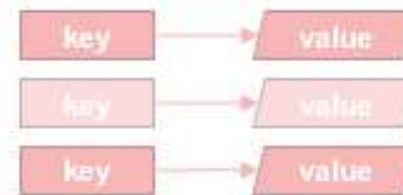
## RELATIONAL

### Row/Column Based

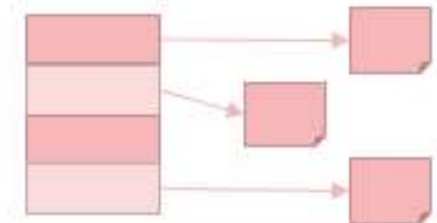


## NON-RELATIONAL

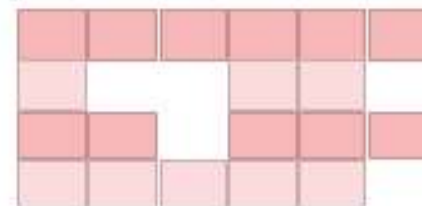
### Key-Value



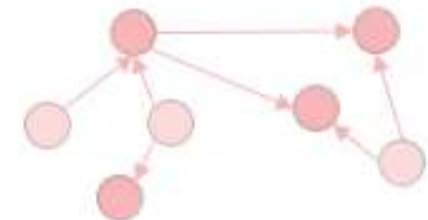
### Document



### Column Family

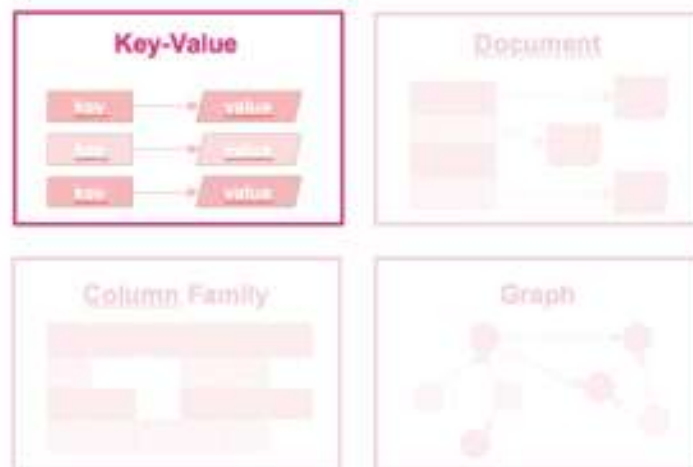


### Graph





# Нереляционная модель данных — ключ-значение



## - Examples:

- Redis,
- BerkeleyDB,
- VoldemortDB,
- ArangoDB,
- Riak, ...

## - Strenghts:

- **fast queries** (value lookups)
- **fast inserts** (key-value pairs)
- **easy to replicate and distribute** (*consistent hashing*)

## - Weaknesses:

- **no or less efficient and slow:**
  - aggregation
  - filtering
  - joining
- needs to be done by application



# Нереляционная модель данных — MapReduce



**MapReduce** = парадигма программирования и соответствующая реализация для параллельной обработки и генерации больших наборов данных, распределенных в кластере.

- первоначально представлено Джеффри Дином и Санджаем Гемаватом (Google Inc.) в 2004 г.
- **MapReduce** не является ни декларативным языком, ни императивным языком программирования, это нечто среднее
- Парадигма основана на указании:
- **map function**, выполняющая фильтрацию и сортировку, в результате промежуточный набор пар ключ/значение объединяет
- **reduce function** все промежуточные значения, связанные с одним и тем же ключом.

Задания **MapReduce** автоматически распараллеливаются и выполняются на нескольких узлах кластера

# Введение в проблемы распределенных систем BIG DATA: маркеры и разметка данных

Основы маркерного обозначения данных, разметка данных по диапазону ключей и хэшу, секционирование вторичных индексов, повторная балансировка и поиск разделов

# Зачем необходимо разбиение или маркер данных ( репликация)?



**Разбиение или маркер данных** — это процесс непрерывного разделения данных на подмножества и распределения их по нескольким узлам в системе данных. Обычно каждая запись или документ в многораздельной системе данных распространяется и напрямую назначается определенному разделу. Разбиение необходимо для:

:Масштабируемость и производительность. Распределение данных по нескольким узлам, например, повышение производительности операций чтения/записи и пропускной способности, поскольку запросы на чтение/запись могут распределяться по нескольким узлам и обрабатываться одновременно. Таким образом, можно распараллелить ввод-вывод (диск), вычислительную мощность (ЦП), а также масштабировать использование памяти, необходимой для выполнения определенной операции над частью набора данных.

**Низкая задержка:** с помощью секционирования можно размещать данные рядом с тем местом, где они используются (пользовательские или потребительские приложения).

**Доступность:** даже если некоторые узлы выходят из строя, только часть данных находится в автономном режиме.

# Репликация VS разбиение(разделения)

	Репликация	Разбиение(разделения)
<b>Хранилище:</b>	копии одних и тех же данных на нескольких узлах	подмножества (разделы) на нескольких узлах
<b>Представление:</b>	избыточность	распределение
<b>Масштабируемость:</b>	параллельный ввод-вывод	потребление памяти
<b>Доступность:</b>	узлы могут взять на себя нагрузку отказавших узлов	сбои узлов затрагивают только части данных
<b>Разные цели, но обычно используются вместе</b>		

# Разбиение — избегайте путаницы в терминах



Чтобы избежать путаницы с термином «раздел» или «разделение», давайте перечислим некоторые другие термины, которые часто используются как синонимы:

- **shards/sharding** - (например, MongoDB, ElasticSearch или RethinkDB)
- **Vnodes/Virtual Nodes** - (например, Riak или Cassandra)
- **region** - (например, HBase)
- **tablet** - (например, BigTable)
- **vBucket** - (например, Couchbase)
- **Разделение** и **репликация** обычно используются вместе, особенно при создании приложений, интенсивно использующих данные, поскольку наборы данных слишком велики для хранения на одном сервере или реплике, и требуются преимущества репликации (например, избыточность, отказоустойчивость или высокая скорость чтения/записи). Этого можно достичь, сохраняя разделы набора данных на нескольких узлах-репликах.

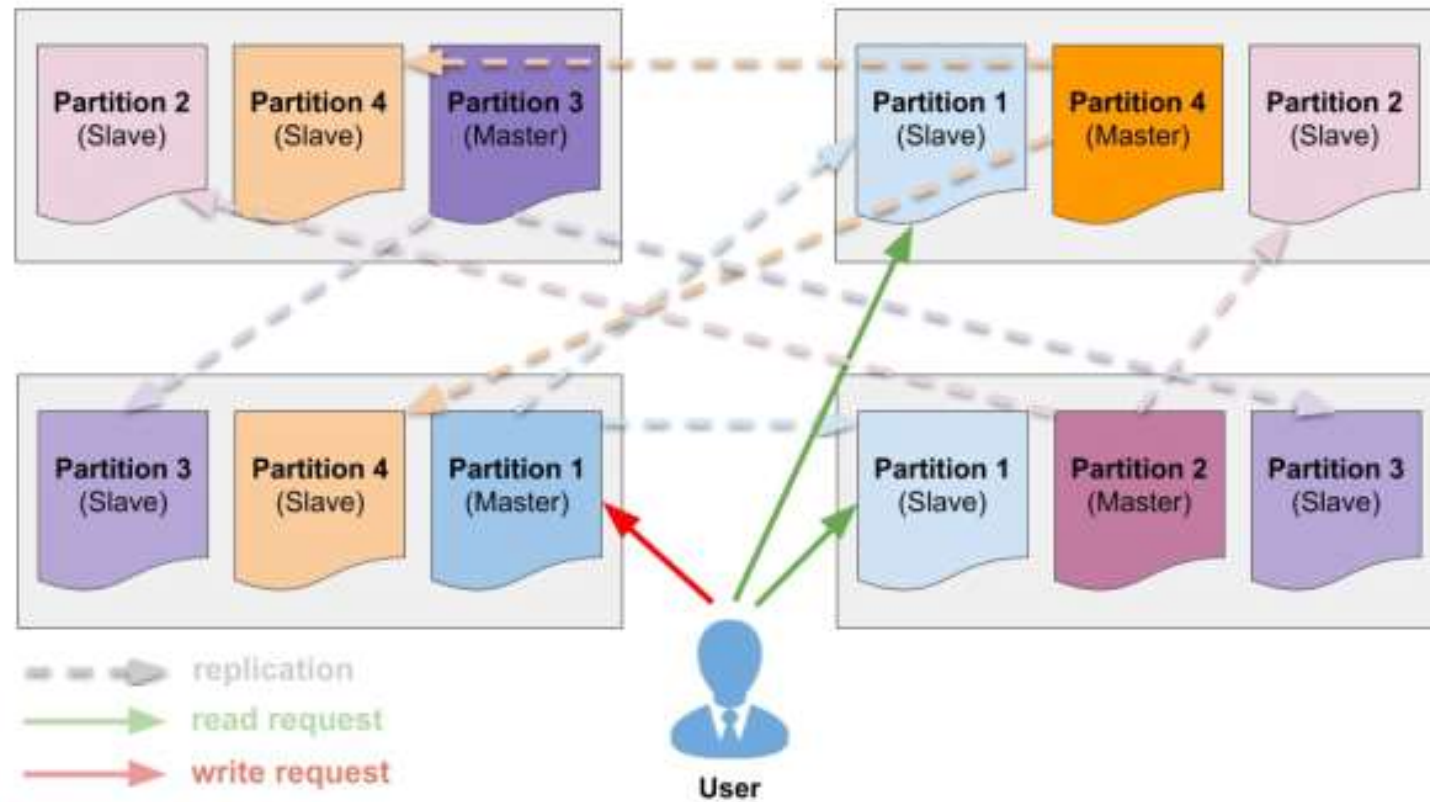
## Разбиение — избегайте путаницы в терминах



Поскольку горизонтальное и вертикальное разбиение иногда объединяют, важно отметить: когда мы говорим о разбиении в рамках нашего предмета - **БОЛЬШИЕ ДАННЫЕ** -, мы имеем в виду **горизонтальное разбиение**.

**Вертикальное разделение** представляет собой подход традиционных реляционных баз данных, обычно выполняемый путем разделения наборов данных на несколько объектов (например, таблиц или баз данных) и использование ссылок (например, для достижения нормализации).

# Разбиение — избегайте путаницы в терминах



## Разбиение и репликация

Использование репликации на основе мастера  
Каждый узел является ведущим для определенного раздела  
Каждый раздел имеет 2 подчиненных узла

**Обеспечение высокой доступности**

# Разбиение — данные типа «Key-Value»

## 2 цели разделения:

- распределение набора данных,
- равномерное распределение связанной нагрузки (запросы на чтение/запись) между несколькими узлами системы данных.

## Это требует:

- правильного способа определения раздела определенной строки или документа что непосредственно влияет на производительность системы данных
  - неправильно выбранный ключ дистрибутива может вызвать:
    - некоторые узлы должны быть бездействующими и/или пустыми и
    - один узел будет узким местом обработки и попадет в его пространство ограничения, так как все запросы на чтение/запись заканчиваются на этом единственном узле



# Разбиение — данные типа «Key-Value»



**Partitioning By Hash Value Of A Key**(Разбиение по хэш-значению ключа):  
получение раздела по определенному хэшу данного ключа для достижения более равномерного распределения данных.

**Partitioning By List**(Разбиение по списку):  
Каждому используемому разделу назначен список значений. Связанный раздел получается из входного набора данных путем проверки, содержит ли он одно из этих значений.

**Round-Robin Partitioning**(Метод циклического перебора):  
очень простой подход, обеспечивающий равномерное распределение данных.  
1-й РЯД пойдет к node1  
2-й РЯД пойдет к node2  
3-й РЯД пойдет к node3  
4-й РЯД пойдет к node1  
5-й РЯД пойдет к node2

# Подготовка к упражнениям

Настройка Сервера HiveServer2

# Pull and Start Docker Container

## 1. Pull Docker Image:

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

## 2. Start Docker Image:

```
sudo 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:

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

## 2. Switch to hadoop user:

```
sudo su hadoop
```

```
cd
```

## 3. Start Hadoop Cluster:

```
start-all.sh
```

# Start HiveServer2

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

```
hive/bin/hiveserver2
```

```
2021-02-21 16:43:55: Starting HiveServer2
```

```
SLF4J: Class path contains multiple SLF4J bindings.
```

```
SLF4J: Found binding in [jar:file:/home/hadoop/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/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 = ae41ac72-4dbd-4115-9863-59c3859c3db6
```

```
Hive Session ID = 17f9f63b-4018-4976-bb7d-15fbf1bc8042
```

```
Hive Session ID = 83b2ad76-c248-46a1-91d4-f2ad289614ee
```

```
Hive Session ID = b9ff1fd3-ccb1-4254-abc7-4c696d8ff8a1
```

```
[...]
```

# Connect To HiveServer2 via JDBC

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

Mac OSX:

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

Linux (Debian):

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

Linux (RPM):

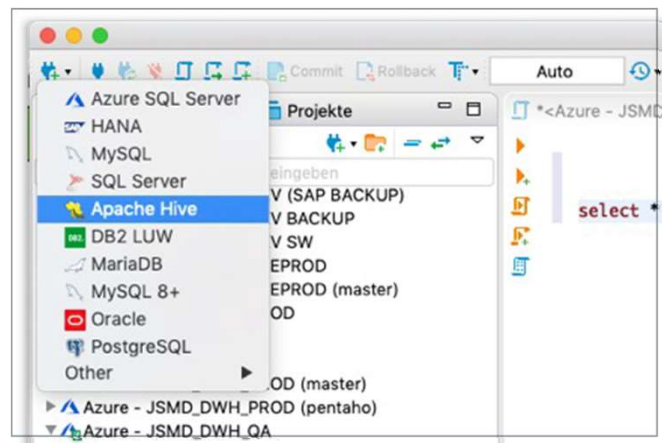
```
wget https://dbeaver.io/files/dbeaver-ce-latest-stable.x86_64.rpm
```

Windows:

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

# Connect To HiveServer2 via JDBC

## 2. Configure Connection To Hive Server:



The 'Neue Verbindung anlegen' (Create New Connection) dialog box for Hadoop / Apache Hive. The 'Allgemeine JDBC Verbindungseinstellungen' (General JDBC Connection Settings) tab is active. The 'JDBC URL' is set to 'jdbc:hive2://XXX.XXX.XXX.XXX:10000/default'. The 'Host' is 'XXX.XXX.XXX.XXX' and the 'Port' is '10000'. The 'Datenbank/Schema' is 'default'. The 'Benutzername' (Username) is 'hadoop' and the 'Passwort' (Password) is empty. The 'Passwort lokal speichern' (Save password locally) checkbox is checked. The 'Treibername' (Driver name) is 'Hadoop / Apache Hive'. The 'Treiber-eigenschaften' (Driver properties) tab is also visible. The 'Advanced settings' section includes 'Netzwerkeinstellungen (SSH, SSL, Proxy, ...)' and 'Verbindungsdetails (Name, Typ, ...)'. The 'Treiber-einstellungen bearbeiten' (Edit driver settings) button is present. The bottom of the dialog has buttons for '?', '< Zurück', 'Weiter >', 'Abbrechen', 'Verbindung testen ...', and 'Fertigstellen'.

# Let's get some data...

## 1. Get some IMDb data:

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

## 2. Put it into HDFS:

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

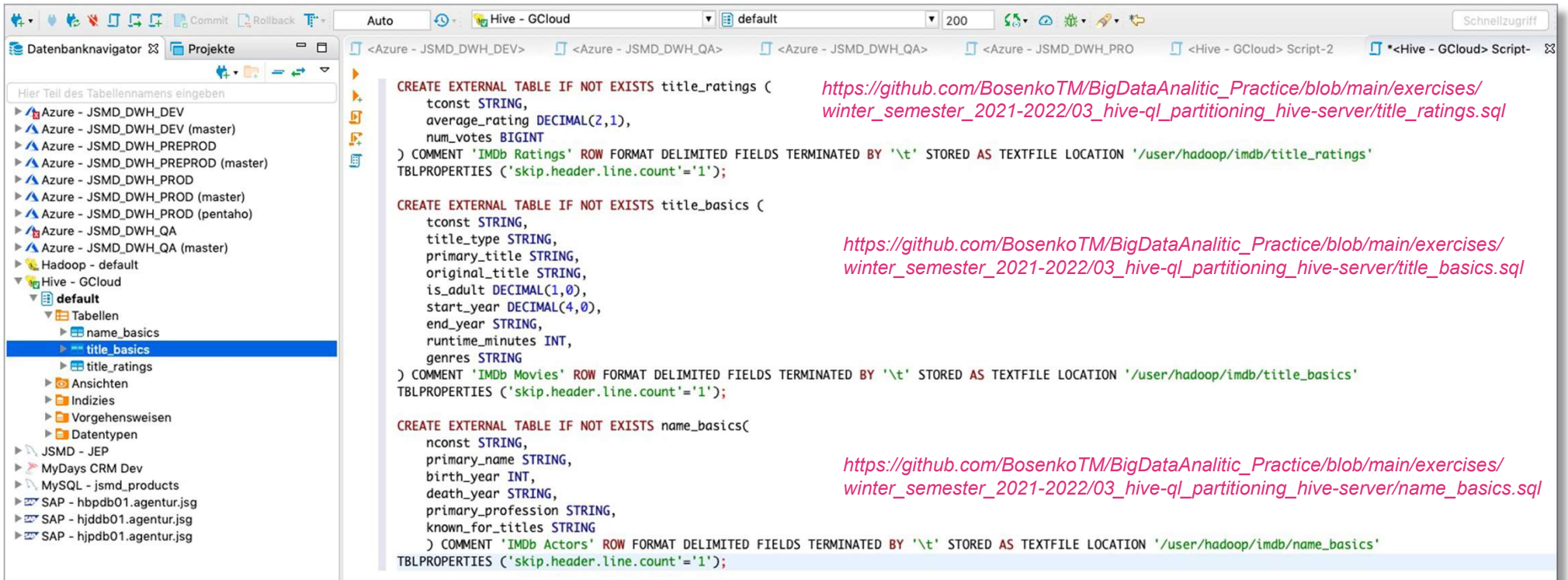
```
hadoop fs -mkdir /user/hadoop/imdb/title_basics && hadoop fs -mkdir /user/hadoop/imdb/title_r  
atings && hadoop fs -mkdir /user/hadoop/imdb/name_basics
```

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



# Create some external tables...

## 1. Create some tables on top of files:



The screenshot shows a Hive IDE interface with a project tree on the left and a SQL editor on the right. The project tree lists various databases and tables, including 'title\_basics' and 'title\_ratings' under the 'default' schema. The SQL editor contains three CREATE EXTERNAL TABLE statements. Each statement is followed by a link to its source file on GitHub.

```
CREATE EXTERNAL TABLE IF NOT EXISTS title_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/title_ratings'  
TBLPROPERTIES ('skip.header.line.count'='1');
```

[https://github.com/BosenkoTM/BigDataAnalytic\\_Practice/blob/main/exercises/winter\\_semester\\_2021-2022/03\\_hive-ql\\_partitioning\\_hive-server/title\\_ratings.sql](https://github.com/BosenkoTM/BigDataAnalytic_Practice/blob/main/exercises/winter_semester_2021-2022/03_hive-ql_partitioning_hive-server/title_ratings.sql)

```
CREATE EXTERNAL TABLE IF NOT EXISTS title_basics (  
  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/title_basics'  
TBLPROPERTIES ('skip.header.line.count'='1');
```

[https://github.com/BosenkoTM/BigDataAnalytic\\_Practice/blob/main/exercises/winter\\_semester\\_2021-2022/03\\_hive-ql\\_partitioning\\_hive-server/title\\_basics.sql](https://github.com/BosenkoTM/BigDataAnalytic_Practice/blob/main/exercises/winter_semester_2021-2022/03_hive-ql_partitioning_hive-server/title_basics.sql)

```
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' STORED AS TEXTFILE LOCATION '/user/hadoop/imdb/name_basics'  
TBLPROPERTIES ('skip.header.line.count'='1');
```

[https://github.com/BosenkoTM/BigDataAnalytic\\_Practice/blob/main/exercises/winter\\_semester\\_2021-2022/03\\_hive-ql\\_partitioning\\_hive-server/name\\_basics.sql](https://github.com/BosenkoTM/BigDataAnalytic_Practice/blob/main/exercises/winter_semester_2021-2022/03_hive-ql_partitioning_hive-server/name_basics.sql)

# 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.average\_rating, r.num\_votes

	tconst	original_title	start_year	average_rating	num_votes
1	tt1375666	Inception	2.010	8,8	2.175.411
2	tt5813916	Dag II	2.016	8,7	106.851
3	tt10295212	Shershaah	2.021	8,7	103.299
4	tt0816692	Interstellar	2.014	8,6	1.620.488
5	tt6751668	Gisaengchung	2.019	8,6	665.500
6	tt1675434	Intouchables	2.011	8,5	798.478
7	tt2582802	Whiplash	2.014	8,5	765.718
8	tt1345836	The Dark Knight Rises	2.012	8,4	1.581.037
9	tt1853728	Django Unchained	2.012	8,4	1.430.287
10	tt7286456	Joker	2.019	8,4	1.072.853

СПАСИБО ЗА ВНИМАНИЕ