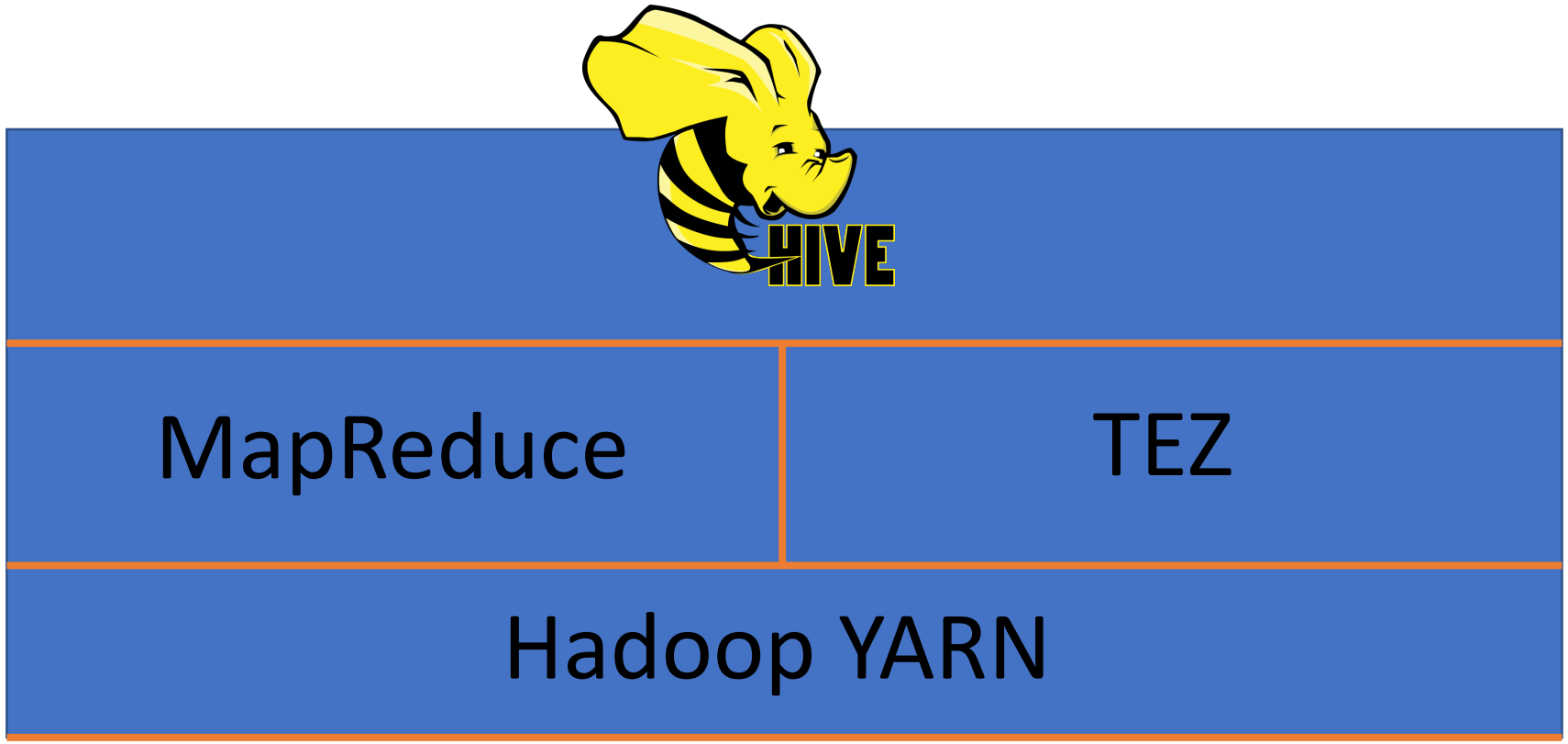
Using relational data stores with Hadoop

Bernard Lee Kok Bang

Hive

- Distributing SQL queries with Hadoop
- Makes Hadoop clusters **looks like** a relational database
- Allows users to write standard SQL queries that look just like when we are using MySQL
- Execute on data that are stored across entire Hadoop clusters

Hive Architecture



- **Translates SQL queries to MapReduce or TEZ jobs on Hadoop clusters**
- **Hive will break down the SQL queries into Mappers and Reducers**

Why Hive

- Use familiar SQL syntax (**HiveQL**)
- Interactive
- Scalable – works with "big data" on a cluster
  →Appropriate for data warehouse application
- Easy Online Analytics processing (OLAP) – much easier than writing MapReduce in Java

OLAP:
- **allows us to look at our data from different perspectives or angles**
- **having a multi-dimensional view of our data**
- **allows us to examine the data based on various factors, such as time, location, product, or customer.**

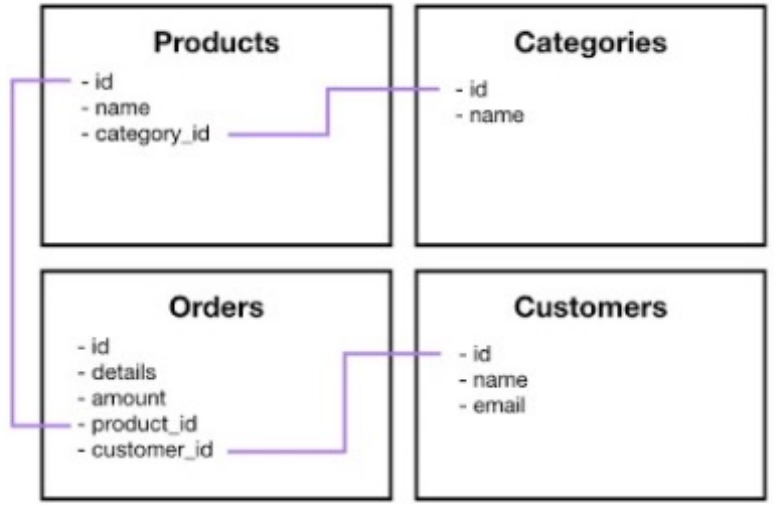
Why not Hive?

- High latency – not appropriate for Online Transaction Processing (OLTP) **[providing real-time transactional processing capabilities]**
  - →Not suitable for “high throughput, low latency” situation
- HDFS stores denormalized data
  - →Hive is not suitable to deal with de-normalized data
- SQL is limited in what it can do
  - →Pig, Spark allows more complex stuff

Normalized vs denormalized data

Normalized

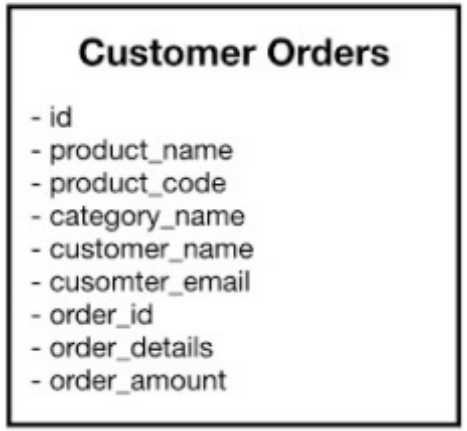
A schema design to store **non-redundant** and **consistent data**



- Data Integrity is maintained
- Little to no redundant data
- Many tables
- Optimizes for storage of data

Denormalized

A schema that **combines data** so that **accessing data (querying) is fast**



- Data Integrity is not maintained
- Redundant data is common
- Fewer tables

https://bit.ly/3vQ91Ax

HiveQL

- Pretty much MySQL with some extensions
- For example: **views**

  →Can store results of a query into a ***"view"***, which subsequent queries can use as a table
- Allows to specify how structured data is stored and partitioned
- HiveQL can do pretty much everything we can do with MySQL

Use Hive to find the most popular movie

- Using Hive to analyze Movielens data (ml-100k dataset)
- First, clear the remnant tables left behind if there are

```
CREATE VIEW topMovieIDs AS
SELECT movieID, count(movieID) as ratingCount
FROM ratings
GROUP BY movieID
ORDER BY ratingCount DESC;

SELECT n.title, ratingCount
FROM topMovieIDs t JOIN names n ON t.movieID = n.movieID;

DROP VIEW topmovieids;
```

[the topmoviesIDs view will still be under the default database]

using u.data -> ratings &
u.item -> names

n.title	ratingcount
Star Wars (1977)	583
Contact (1997)	509
Fargo (1996)	508
Return of the Jedi (1983)	507
Liar Liar (1997)	485
English Patient, The (1996)	481
Scream (1996)	478
Toy Story (1995)	452
Air Force One (1997)	431
Independence Day (ID4) (1996)	429

Schema on Write (Traditional Databases)

- used by *traditional relational databases* like MySQL, PostgreSQL, Oracle

How it works:

- The ***schema*** (table structure, data types, constraints) is defined and enforced ***before*** data is written to the database
- Data must *conform to the predefined schema* when it's inserted
- If data doesn't match the schema, the *write operation fails*
- Data validation and type checking happens at write time

Schema on Read (Hive Approach)

- used by Hive and other big data systems

How it works:

- Data is stored in its ***raw format*** (CSV, JSON, Parquet, etc.) without enforcing a schema
- The schema is applied ***when*** the data is read/queried, not when it's written
- Data validation and interpretation happens at query time
- Provides flexibility to reinterpret the same data with different schemas

Example in Hive:

```sql
-- Data already exists in HDFS as a CSV file: /data/employees.csv
-- File contains: 1,John Doe,50000.00,2023-01-15

-- Create external table that maps to existing data
CREATE EXTERNAL TABLE employees (
    id INT,
    name STRING,
    salary DOUBLE,
    hire_date STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION '/data/';

-- Schema is applied when you query, not when data was written
SELECT * FROM employees WHERE salary > 40000;
```

Key Differences in Hive Context

Aspect	Schema on Write	Schema on Read (Hive)
Data Loading	Must validate against schema first	Data loaded as-is, no validation
Performance	Faster queries (pre-validated)	Slower queries (validation at runtime)
Flexibility	Low - schema changes are expensive	High - can reinterpret same data differently
Data Quality	High - bad data rejected upfront	Variable - bad data discovered at query time
Storage	Structured, optimized format	Raw format, multiple formats supported

Find the movie with the highest average rating

- **we are creating a "view"**
- **"views" are persistent, and stored to disk**
- **will get an error if the same "view" is created more than once**

- *AVG()* can be used on aggregated data like *COUNT()* does.
- Consider only movies with more than 10 ratings

```
CREATE VIEW IF NOT EXISTS avgRatings AS
SELECT movieID, AVG(rating) as avgRating, COUNT (movieID) as ratingCount
FROM ratings
GROUP BY movieID
ORDER BY avgRating DESC;

SELECT n.title, avgRating
FROM avgRatings t JOIN names n ON t.movieID = n.movieID
WHERE ratingCount > 10;
```

n.title	avgrating
Close Shave, A (1995)	4.491071428571429
Schindler's List (1993)	4.466442953020135
Wrong Trousers, The (1993)	4.466101694915254
Casablanca (1942)	4.45679012345679
Wallace & Gromit: The Best of Aardman Animation (1996)	4.447761194029851
Shawshank Redemption, The (1994)	4.445229681978798
Rear Window (1954)	4.3875598086124405
Usual Suspects, The (1995)	4.385767790262173
Star Wars (1977)	4.3584905660377355

Hive’s view

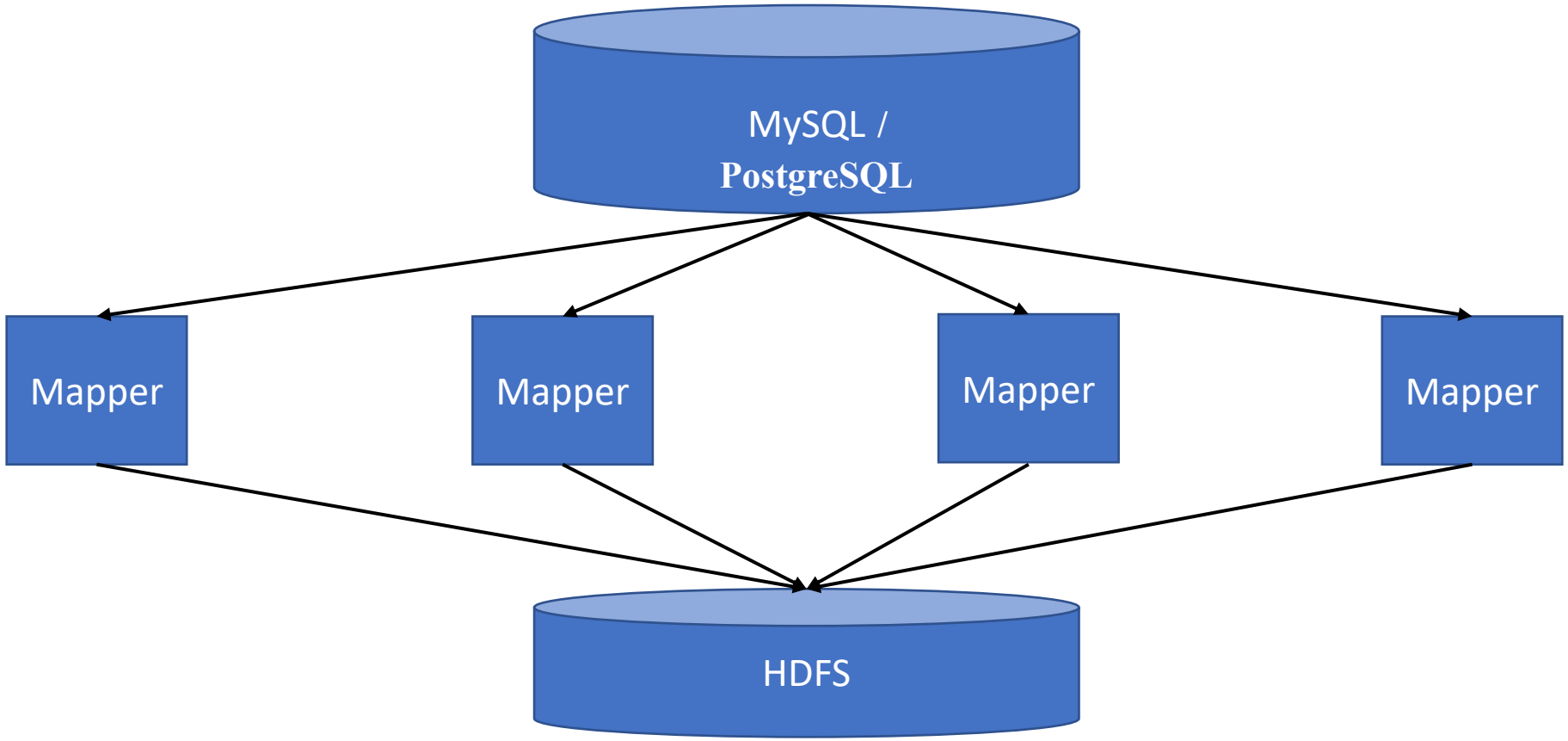
- a view in Hive is like a virtual book that represents a specific subset of your data.
- It's a saved query that provides a customized perspective without duplicating the data itself.
- It offers convenience, abstraction, and a way to quickly access the information you need from your dataset.

What's MySQL

- Popular, free relational database [tables with rows and columns]
- Generally monolithic in nature [e.g. stored in single server with giant hard drive; limited in what we can do because it is not distributed across a cluster]
- But, can be used for OLTP – so exporting data into MySQL can be useful
- Existing data may exist in MySQL that you want to import into Hadoop
- Using ***Sqoop*** [*SQ*L + Had*oop*]

Sqoop can handle BIG data

- Kicks off MapReduce jobs to handle importing or exporting data



Let’s play with MySQL and Sqoop

- Import Movielens data into MySQL database
- Import the movies to HDFS **[from MySQL -> HDFS]**
- Import the movies into Hive **[from MySQL -> Hive]**
- Export the movies back into MySQL **[from Hive to MySQL]**

Need to set up default password for MySQL on HDP 2.65

su root ~~[default password is hadoop]~~
~~(current) UNIX password: hadoop~~
~~New password: xxxxx~~
~~Retype new password: xxxxx~~
systemctl stop mysqld
systemctl set-environment MYSQLD_OPTS="--skip-grant-tables --skip-networking"
systemctl start mysqld
mysql -uroot

------- Mysql cmd
FLUSH PRIVILEGES;
alter user 'root'@'localhost' IDENTIFIED BY 'hadoop';
FLUSH PRIVILEGES;
QUIT;

in MySQL shell

------ CMD
systemctl unset-environment MYSQLD_OPTS
systemctl restart mysqld
exit

Import data into MySQL [password for MySQL is hadoop]

```
# Download the sql script
wget http://media.sundog-soft.com/hadoop/movielens.sql

# key in credentials to log into mysql
mysql -u root -p
hadoop

# Creating ~~tables~~ database
create database movielens;
show databases;

# Set some environment
SET NAMES 'utf8';
SET CHARACTER SET utf8;

# Start the importing process
use movielens;
source movielens.sql;
show tables;
select * from movies limit 10;

describe ratings;

# Find most popular movies using MySQL
SELECT movies.title, COUNT (ratings.movie_id) AS ratingCount
FROM movies
INNER JOIN ratings
ON movies.id = ratings.movie_id
GROUP BY movies.title
ORDER BY ratingCount;

exit
```

Import data from MySQL to HDFS [In Ambari Files View]

- First need to set the appropriate permissions on MySQL so that Sqoop can access

use only one process

```
mysql -u root -p
hadoop
GRANT ALL PRIVILEGES on movielens.* to root@localhost identified by 'hadoop';
exit
# On command prompt
sqoop import --connect jdbc:mysql://localhost/movielens --driver com.mysql.jdbc.Driver --table movies -m 1
--username root --password hadoop
```

where to find the data

use which software to 'talk' to MySQL

our credential

- Navigate back to Ambari to check for the imported data [under maria_dev folder]
- **Remember to clean up the created data once finished with the current session**

To specify destination directory, add this:
--target-dir /user/maria_dev/bernard/movies

Import data from MySQL to Hive

```
# On command promt
sqoop import --connect jdbc:mysql://localhost/movielens --driver com.mysql.jdbc.Driver --table movies -m 1
--username root --password hadoop --hive-import
```

- Navigate back to Ambari (Hive View) to check for the imported data

Export data from Hive to MySQL

- First need to locate where the data in Hive resides
- Hive is just a **schema on read**
- The actual data is stored as a plain text file in another location [*/apps/hive/warehouse/movies/part-m-00000*]
- Need to make sure the table exists ahead of time in MySQL before exporting data from Hive to MySQL

```
mysql -u root -p
hadoop
use movielens;
CREATE TABLE exported_movies (id INTEGER, title VARCHAR (255), releaseDate DATE);

exit
```

Export data from Hive to MySQL (cont)

sqoop export --connect jdbc:mysql://localhost/movielens -m 1 --driver com.mysql.jdbc.Driver --table exported_movies --export-dir /apps/hive/warehouse/movies --input-fields-terminated-by '\001' --username root --password hadoop

 # At command prompt
mysql -u root -p
hadoop

use movielens
*SELECT * from exported_movies limit 10;*