

# Hive: SQL for Hadoop

An Essential Tool for  
Hadoop-based  
Data Warehouses



Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, January 10, 12

I'll argue that Hive is indispensable to people creating "data warehouses" with Hadoop, because it gives them a "similar" SQL interface to their data, making it easier to migrate skills and even apps from existing relational tools to Hadoop.

# Adapted from our *Hadoop 3-Day Developer Course*

[info@thinkbiganalytics.com](mailto:info@thinkbiganalytics.com)

[thinkbiganalytics.com](http://thinkbiganalytics.com)



Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, January 10, 12

These notes are excerpts from our Hadoop 3-day developer course. The second day focuses on Hive. We also offer Hive training by itself, admin courses, etc.

# *Programming Hive* Summer, 2012 O'Reilly



Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, January 10, 12

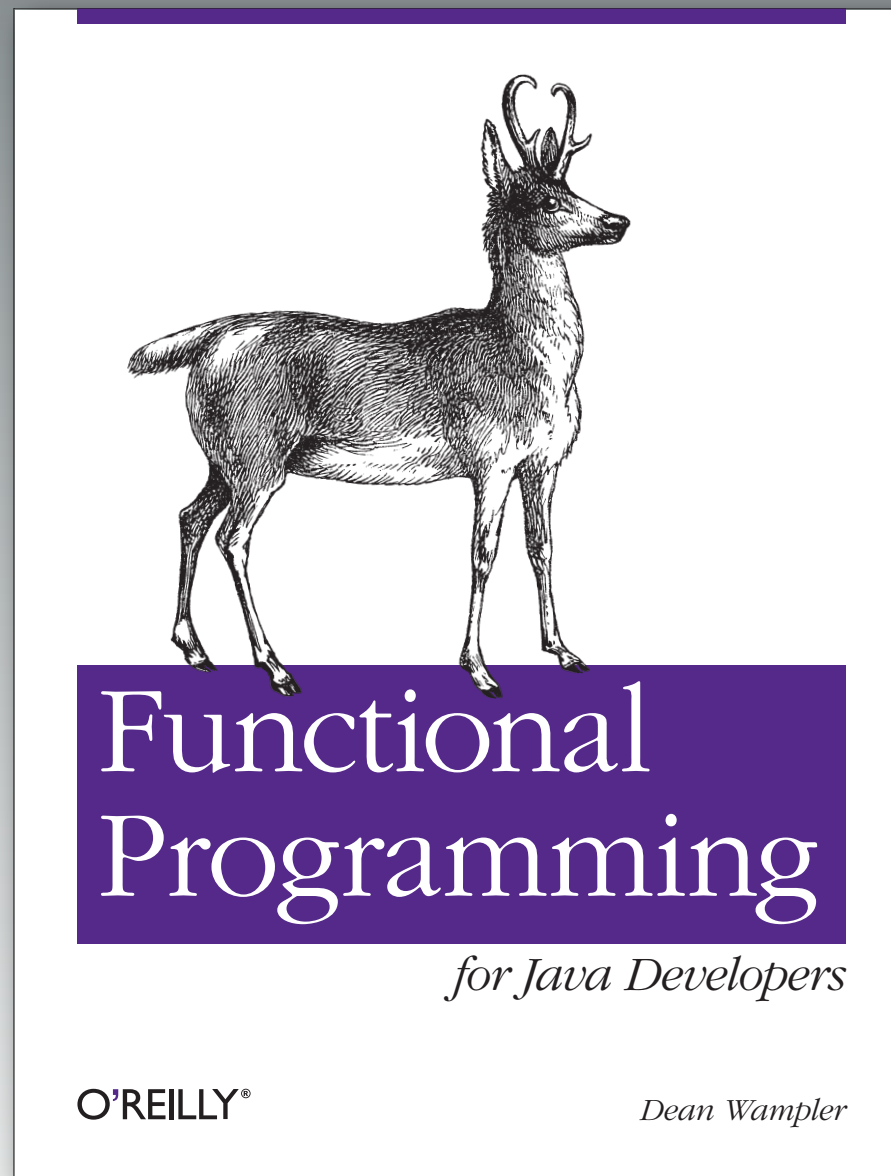
I'm collaborating with two other colleagues to write "Programming Hive", which will be published by O'Reilly this summer. Until it's ready, the best source of documentation is on the Hive site, <http://hive.apache.org>.



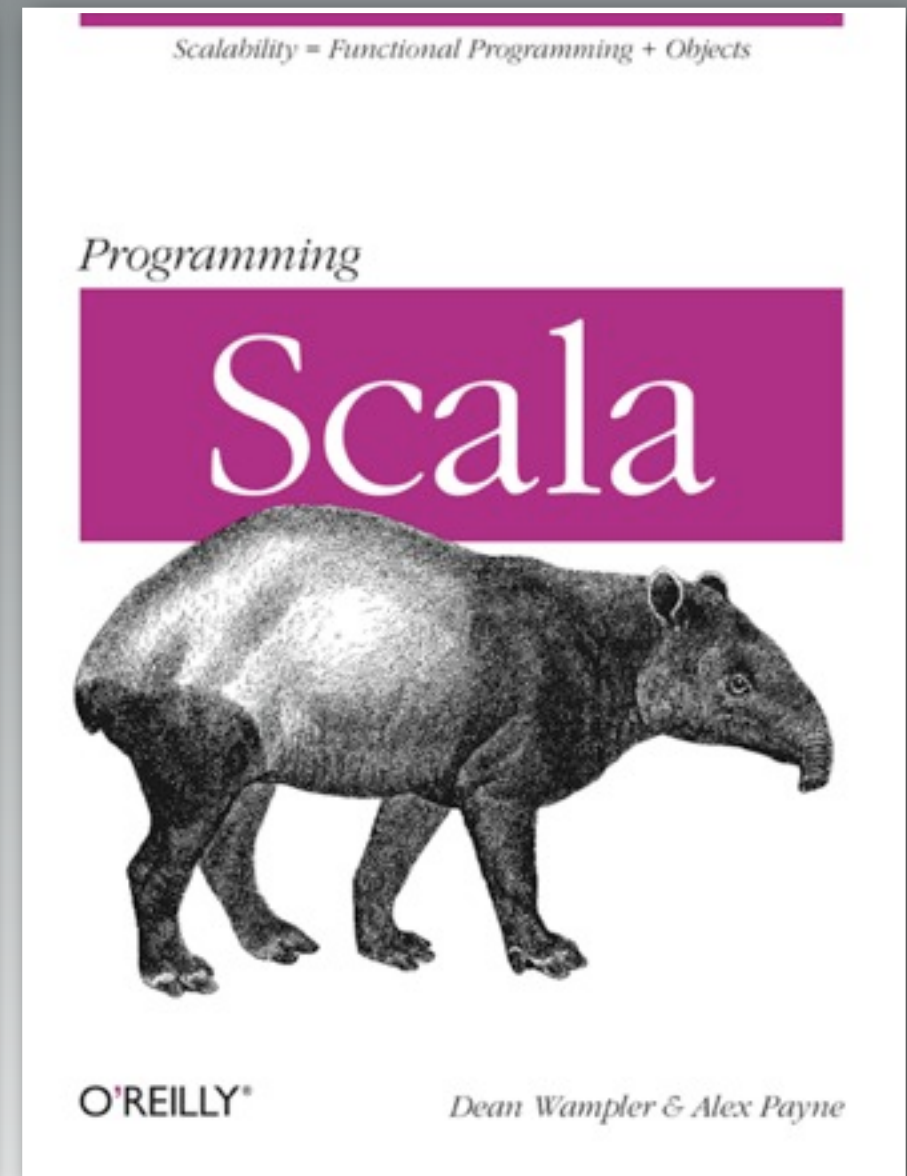
# Dean Wampler

20+ years exp.

*Think Big Academy*



[polyglotprogramming.com/fpjava](http://polyglotprogramming.com/fpjava)



[programmingscala.com](http://programmingscala.com)

# A True Story...



Once upon an Internet  
time, in the land of  
Enterprise, the King's  
Warehouse of Data was  
bursting at the seams.



And lo, the scribes  
decided they could cut  
costs and store more  
data by moving to  
Hadoop.



But then the scribes  
complained amongst  
themselves, "The  
language of the land of  
Java hurts our tongues!



“It takes forever to say  
anything!”



There was a scribe who  
walked through the  
Kingdom-wide Labyrinth  
(kwl), where he found  
The Book of Faces.



In that book, he found  
the secret language the  
Bee Keepers use to  
query the god Hadoop!



And there was much  
rejoicing, for the scribes  
could work with their  
data again!



The End.



# Enter *Hive*



Since your team *knows SQL*  
and all your *Data Warehouse*  
apps are written in *SQL*,  
*Hive* minimizes the effort of  
migrating to *Hadoop*.



# Hive

- Ideal for *data warehousing*.
- *Ad-hoc queries* of data.
- Familiar *SQL* dialect.
- Analysis of *large* data sets.
- Hadoop *MapReduce* jobs.



# Hive

- Invented at *Facebook*.
- Open sourced to *Apache* in 2008.
- <http://hive.apache.org>



# A Scenario: *Mining Daily Click Stream Logs*



# Ingest & Transform:

- **From:** `file://server1/var/log/clicks.log`

```
Jan  9 09:02:17 server1 movies[18]:  
1234: search for “vampires in love”.  
...
```



# Ingest & Transform:

- **From:** `file://server1/var/log/clicks.log`

```
Jan  9 09:02:17 server1 movies[18]:  
1234: search for “vampires in love”.
```

...



Timestamp



# Ingest & Transform:

- **From:** `file://server1/var/log/clicks.log`

```
Jan  9 09:02:17 server1 movies[18]:  
1234: search for "vampires in love".
```

...



The server



# Ingest & Transform:

- **From:** `file://server1/var/log/clicks.log`

Jan 9 09:02:17 server1 **movies[18]**:  
1234: search for “vampires in love”.

...

The process  
 (“movies search”)  
and the process id.



# Ingest & Transform:

- **From:** `file://server1/var/log/clicks.log`

`Jan 9 09:02:17 server1 movies[18]:`

`1234: search for “vampires in love”.`

...

Customer id



# Ingest & Transform:

- **From:** `file://server1/var/log/clicks.log`

```
Jan  9 09:02:17 server1 movies[18]:  
1234: search for “vampires in love”.  
...
```

The log “message”



# Ingest & Transform:

- **From:** `file://server1/var/log/clicks.log`

```
Jan  9 09:02:17 server1 movies[18]:  
1234: search for "vampires in love".  
...
```

- **To:** `/staging/2012-01-09.log`

```
09:02:17^Aserver1^Amovies^A18^A1234^Asea  
rch for "vampires in love".  
...
```

# Ingest & Transform:

- **To:** `/staging/2012-01-09.log`

`09:02:17^Aserver1^Amovies^A18^A1234^Asearch for “vampires in love”.`

...

- *Removed* month (`Jan`) and day (`09`).
- Added `^A` as *field* separators (Hive convention).
- Separated process *id* from process *name*.



# Ingest & Transform:

- Put in HDFS:

```
hadoop fs -put /staging/2012-01-09.log \  
/clicks/2012/01/09/log.txt
```

- (The final file name doesn't matter...)

# Back to Hive...

- Create an *external* Hive table:

```
CREATE EXTERNAL TABLE clicks (  
  hms          STRING,  
  hostname     STRING,  
  process      STRING,  
  pid          INT,  
  uid          INT,  
  message      STRING)  
PARTITIONED BY (  
  year         INT,  
  month        INT,  
  day          INT);
```

You don't have to  
use EXTERNAL  
and PARTITIONED  
together....

Now let's create an "external" table that will read those files as the "backing store". Also, we make it partitioned to accelerate queries that limit by year, month or day. (You don't have to use external and partitioned together...)



# Back to Hive...

- Add a *partition* for 2012-01-09:

```
ALTER TABLE logs ADD IF NOT EXISTS  
PARTITION (  
    year=2012,  
    month=01,  
    day=09)  
LOCATION '/clicks/2012/01/09';
```

- A *directory* in HDFS.

# Now, Analyze!!

- What's with the *kids* and *vampires*??

```
SELECT hms, uid, message FROM clicks
WHERE message LIKE '%vampire%' AND
      year    = 2012 AND
      month   = 01   AND
      day     = 09;
```

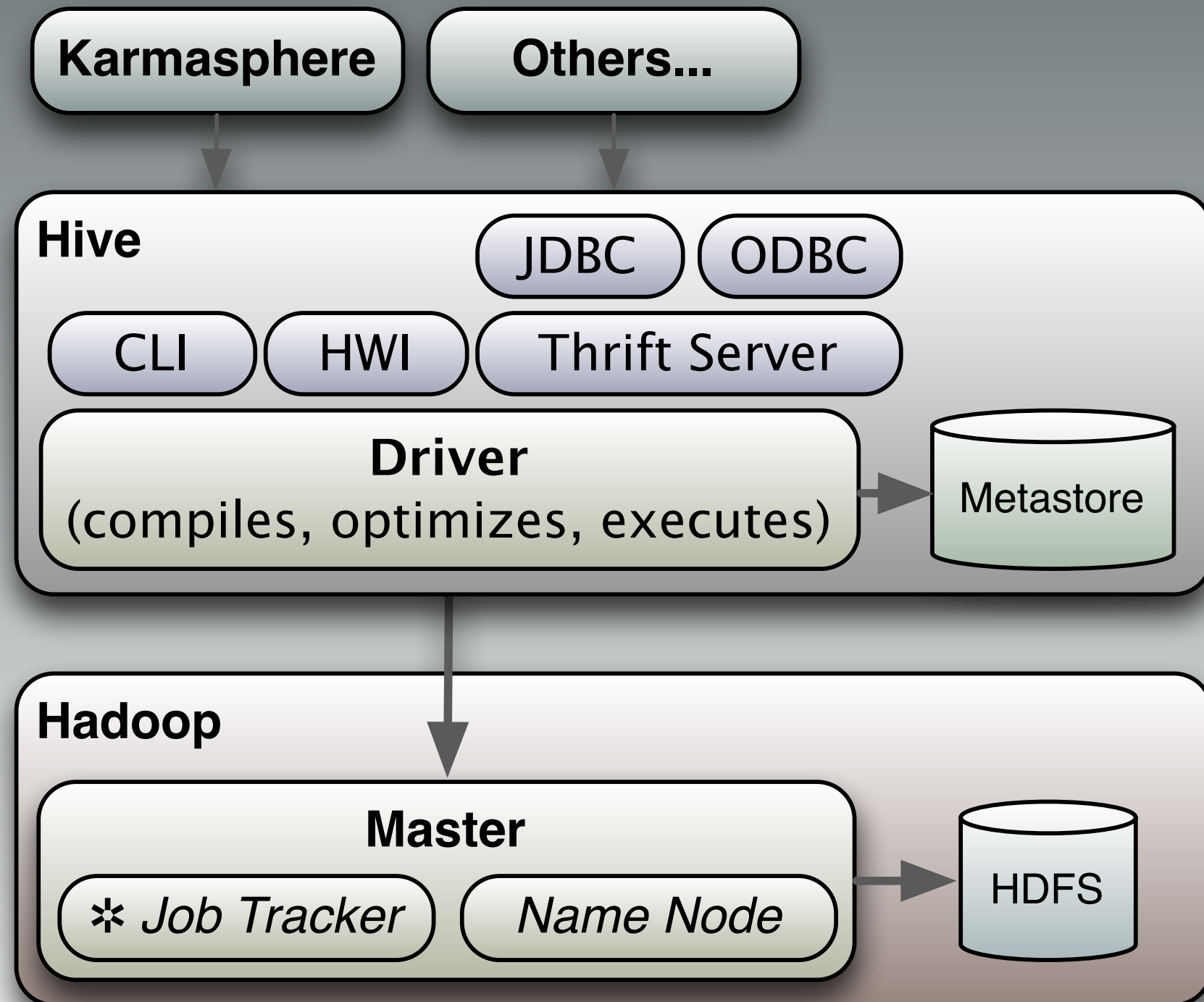
- After some *MapReduce* crunching...

```
...
09:02:29 1234 search for "twilight of the vampires"
09:02:35 1234 add to cart "vampires want their genre back"
...
```



# Recap

- *SQL analysis* with Hive.
- *Other tools* can use the data, too.
- *Massive scalability* with Hadoop.



Hive queries generate MR jobs. (Some operations don't invoke Hadoop processes, e.g., some very simple queries and commands that just write updates to the metastore.)

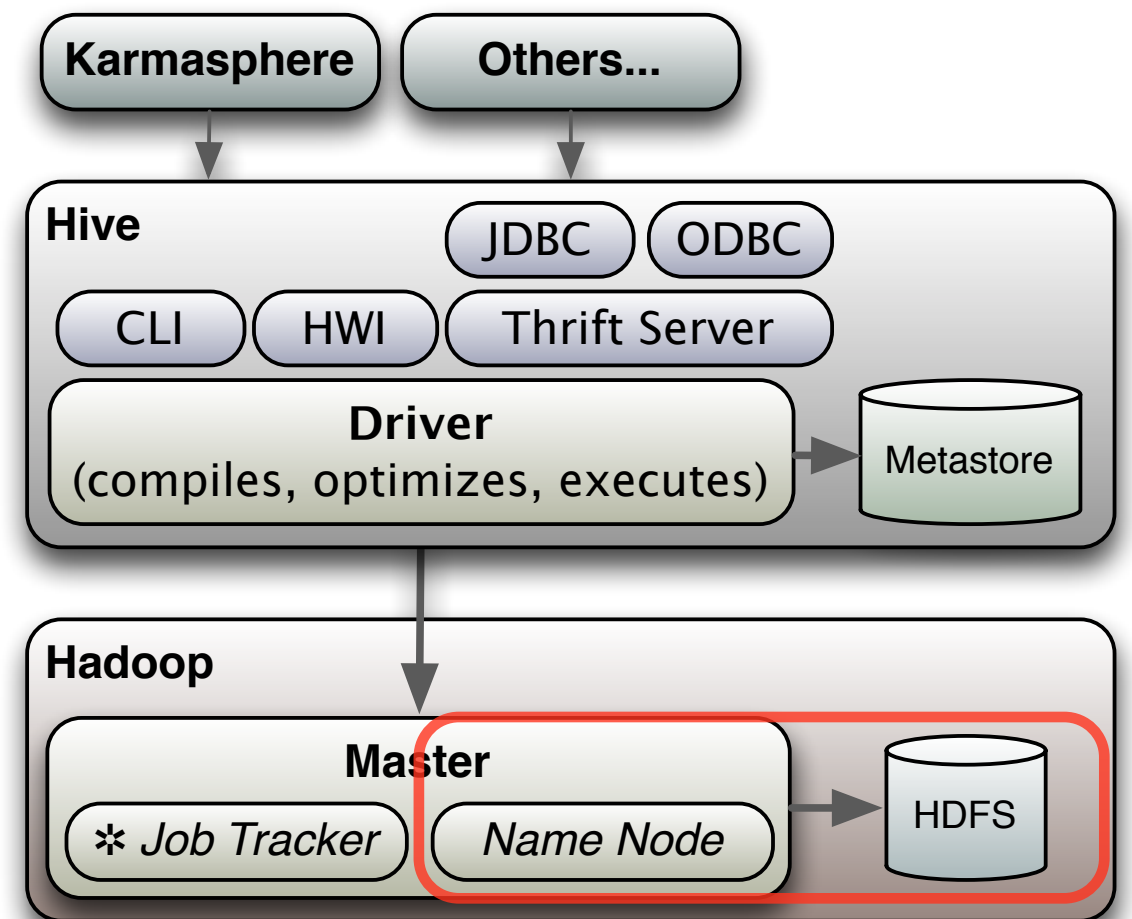
CLI = Command Line Interface.

HWI = Hive Web Interface.



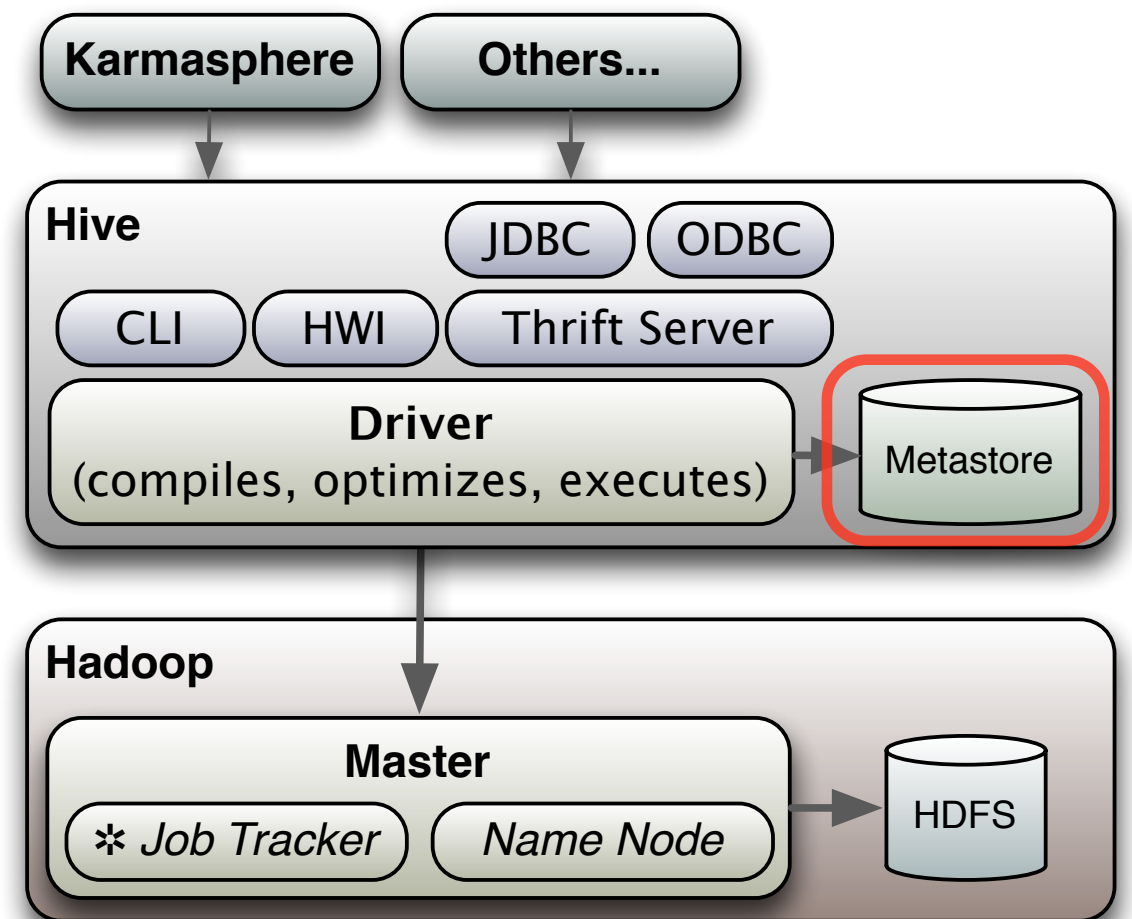
# Tables

- *HDFS*
- *MapR*
- *S3*
- *HBase* (new)
- *Others...*



# Tables

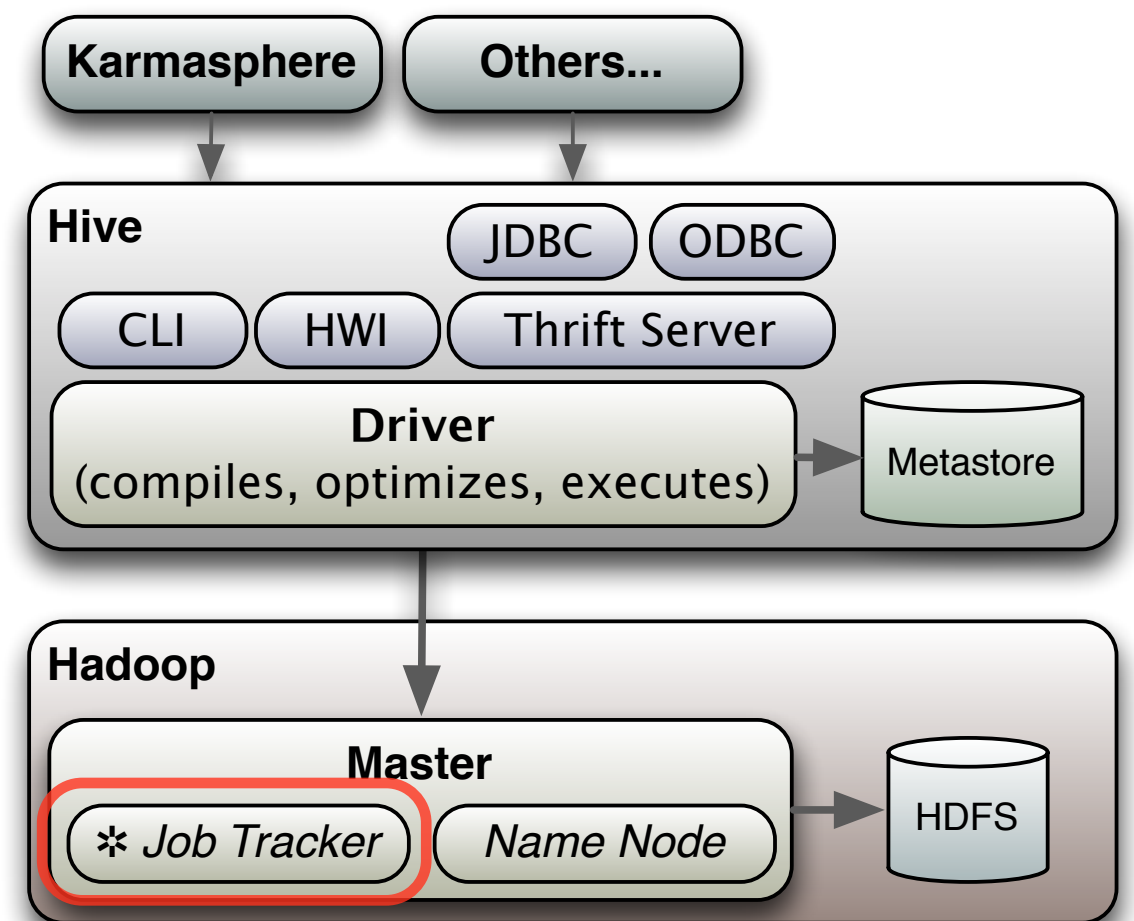
- Table *metadata* stored in a *relational DB*.





# Queries

- Most queries use *MapReduce* jobs.



# MapReduce Queries

- *Benefits*
  - Horizontal scalability.
- *Drawbacks*
  - Latency!



# HDFS Storage

- *Benefits*
  - Horizontal scalability.
  - Data redundancy.
- *Drawbacks*
  - No *insert, update, and delete!*

# HDFS Storage

- *Schema on Read*
- Schema enforcement at *query* time, not *write* time.



# Other Limitations

- No *Transactions*.
- Some *SQL* features not implemented (yet).

# Most on Tables and Schemas



# Data Types

- The usual *scalar* types:
  - TINYINT,... BIGINT.
  - FLOAT, DOUBLE.
  - BOOLEAN.
  - STRING.

Recall our first exercise where we generated Word Count output for Shakespeare's plays. With that output in HDFS, here's how we could create an "external" Hive table that allows us to query that data.

What about sequence files? We'll come back to that shortly...

# Data Types

- The unusual *complex* types:
  - STRUCT.
  - MAP.
  - ARRAY.

Recall our first exercise where we generated Word Count output for Shakespeare's plays. With that output in HDFS, here's how we could create an "external" Hive table that allows us to query that data.

What about sequence files? We'll come back to that shortly...

```
CREATE TABLE employees (  
  name STRING,  
  salary FLOAT,  
  subordinates ARRAY<STRING>,  
  deductions MAP<STRING, FLOAT>,  
  address STRUCT<  
    street:STRING,  
    city:STRING,  
    state:STRING,  
    zip:INT>  
) ;
```

subordinates references other records by the employee name. (Hive doesn't have indexes, in the usual sense, but an indexing feature was recently added.) Deductions is a key-value list of the name of the deduction and a float indicating the amount (e.g., %). Address is like a "class", "object", or "c-style struct", whatever you prefer.



# File & Record Formats

```
CREATE TABLE employees (...)  
  ROW FORMAT DELIMITED  
    FIELDS TERMINATED BY '\001'  
    COLLECTION ITEMS TERMINATED BY '\002'  
    MAP KEYS TERMINATED BY '\003'  
    LINES TERMINATED BY '\n'  
  STORED AS TEXTFILE;
```

All the  
defaults for  
text files!

# Select, Where, Group By, Join,...

# Common SQL...

- You get most of the usual suspects for `SELECT`, `WHERE`, `GROUP BY` and `JOIN`.



# “User Defined Functions”

```
ADD JAR MyUDFs.jar;
```

```
CREATE TEMPORARY FUNCTION net_salary  
AS 'com.example.NetCalcUDF';
```

```
SELECT name, net_salary(salary, deductions)  
FROM employees;
```

# ORDER BY vs. SORT BY

- A *total* ordering - one reducer.

```
SELECT name, salary FROM employees  
ORDER BY salary ASC;
```

- A *local* ordering - sorts within each reducer.

```
SELECT name, salary FROM employees  
SORT BY salary ASC;
```

# Inner Joins

- Only *equality* ( $x = y$ ).

```
SELECT ...  
FROM clicks a  
JOIN clicks b ON (  
    a.uid = b.uid,  
    a.day = b.day)  
WHERE a.process = 'movies'  
      AND b.process = 'books'  
      AND a.year > 2012;
```



# A Final Example of Controlling MapReduce...

# Specify Map & Reduce Processes

- Calling out to *external programs* to perform *map* and *reduce* operations.

# Example

```
FROM (  
  FROM clicks  
  MAP message  
  USING '/tmp/vampire_extractor'  
  AS item_title, count  
  CLUSTER BY item_title) it  
INSERT OVERWRITE TABLE vampire_stuff  
  REDUCE it.item_title, it.count  
  USING '/tmp/thing_counter.py'  
  AS item_title, counts;
```



# Example

```
FROM (  
  FROM clicks  
  MAP message  
  USING '/tmp/vampire_extractor'  
  AS item_title, count  
  CLUSTER BY item_title) it  
INSERT OVERWRITE TABLE vampire stuff  
  REDUCE it.item_title, it.count  
  USING '/tmp/thing_counter.py'  
  AS item_title, counts;
```

Call specific  
map and  
reduce  
processes.

# ... And Also:

```
FROM (  
  FROM clicks  
  MAP message  
  USING '/tmp/vampire_extractor'  
  AS item_title, count  
  CLUSTER BY item_title) it  
INSERT OVERWRITE TABLE vampire_stuff  
  REDUCE it.item_title, it.count  
  USING '/tmp/thing_counter.py'  
  AS item_title, counts;
```

Like GROUP BY, but directs output to specific reducers.

# ... And Also:

```
FROM (  
  FROM clicks  
  MAP message  
  USING '/tmp/vampire_extractor'  
  AS item_title, count  
  CLUSTER BY item_title) it  
INSERT OVERWRITE TABLE vampire_stuff  
  REDUCE it.item_title, it.count  
  USING '/tmp/thing_counter.py'  
  AS item_title, counts;
```

How to  
populate an  
“internal”  
table.



# Hive: Conclusions

# Hive Disadvantages

- *Not a real SQL Database.*
- Transactions, updates, etc.
- ... but features will grow.
- *High latency* queries.
- Documentation poor.

# Hive Advantages

- Indispensable for *SQL users*.
- *Easier* than *Java* MR API.
- Makes *porting* data warehouse apps to Hadoop much *easier*.



# Questions?

[dean.wampler@thinkbiganalytics.com](mailto:dean.wampler@thinkbiganalytics.com)

[thinkbiganalytics.com](http://thinkbiganalytics.com)

**This presentation:** [github.com/deanwampler/Presentations](https://github.com/deanwampler/Presentations)