MITS6005

Big Data

VICTORIAN
INSTITUTE OF
TECHNOLOGY
www.vit.edu.au

Session 8

Hive and Pig

Overview – last session

- HBase?
- HBase – features
- HBase vs HDFS
- HBase vs RDBMS
- Practical examples

Overview – this week

- Need for High-Level Languages in Big Data
- What is Hive?
- Why Apache Hive?
- Hive vs MapReduce
- Hive Architecture
- Features of Apache Hive
- Limitation of Apache Hive
- Practical points

Need for High-Level Languages

- Hadoop is great for large-data processing!
  - But writing Java programs for everything is verbose and slow
  - Not everyone wants to (or can) write Java code

- Which language should we use?
  - Java: Java Data Mining Package (JDMP) is a Java library for machine learning and Big Data Analytics
  - Python: most big data processing frameworks do support Python; pydoop and PySpark
  - Scala: Kafka and Spark is powered by Scala
  - The Hadoop **languages** (Pig, Hive, etc.)

- higher-level data processing languages
  - Hive: HiveQL is like SQL
  - Pig: Pig Latin is a bit like Perl

Hive: Introduction

- Hive evolved as a data warehousing solution built on top of Hadoop Map-Reduce framework.
- ETL tool for Hadoop ecosystem (e.g. Apache Hive on HDInsight or hand-coded python).
- Hive provides SQL-like declarative language, called **HiveQL,** for MapReduce jobs which requires writing complex codes.
- **Hive engine** compiles these queries into Map-Reduce jobs to be executed on Hadoop.
- Hive comes with a command-line shell interface to create tables and execute queries.
- Hive query makes it possible to take a MapReduce joins across Hive tables. It has a support for simple **SQL like functions** - CONCAT, SUBSTR, ROUND etc., and **aggregation functions**- SUM, COUNT, MAX etc. It also supports GROUP BY and SORT BY clauses.

Appache Hive

- Started at Facebook.
- Apache Hive: a data warehouse framework for querying and managing large data that is stored in HDFS.
- Apache Hive-Hadoop cluster at Facebook stores more than 2PB of raw data. It regularly loads 15 TB of data on a daily basis.
- It is being used and developed by a number of companies like Amazon, IBM, Yahoo, Netflix, Financial Industry Regulatory Authority (FINRA) and many others

Why Appache Hive?

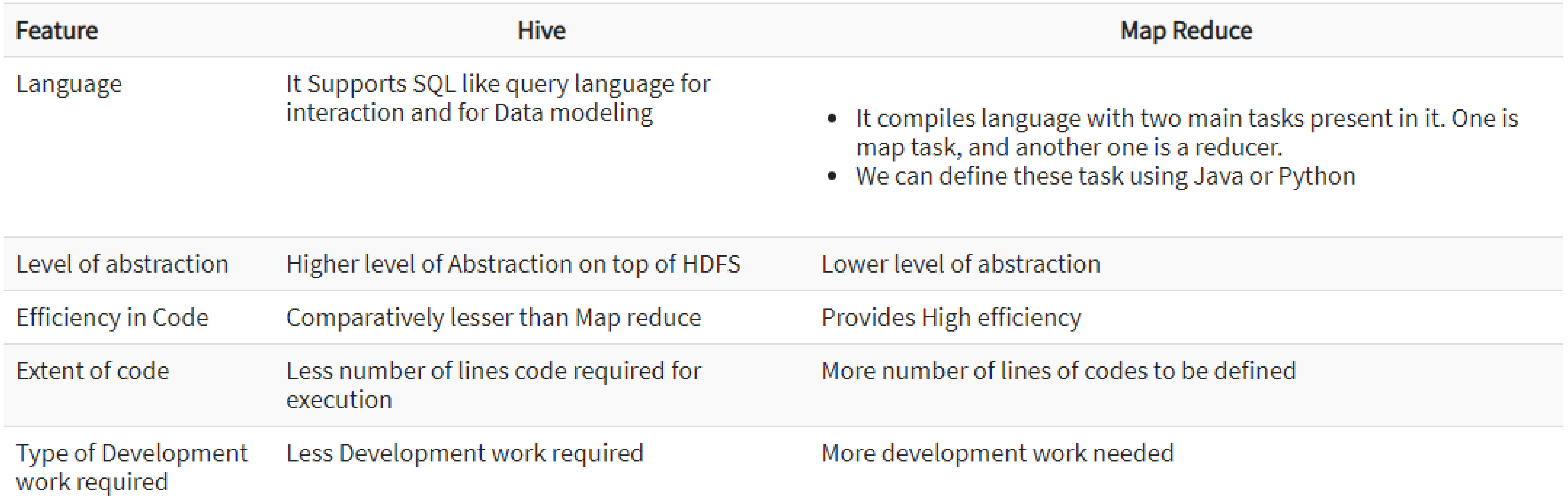
- Apache Hive saves developers from writing complex Hadoop MapReduce jobs for ad-hoc requirements.
- Hive provides summarization, analysis, and query of data.
- Hive is very fast and scalable. It is highly extensible.
- Since Apache Hive is similar to SQL, hence it becomes very easy for the SQL developers to learn and implement Hive Queries.
- It also provides file access on various data stores like HDFS and HBase. The most important feature of Apache Hive is that to learn Hive we don't have to learn Java.

https://data-flair.training/blogs/apache-hive-tutorial/

Hive vs MapReduce

While choosing between Hive and MapReduce following factors are taken in consideration;

- Type of Data
- Amount of Data
- Complexity of Code

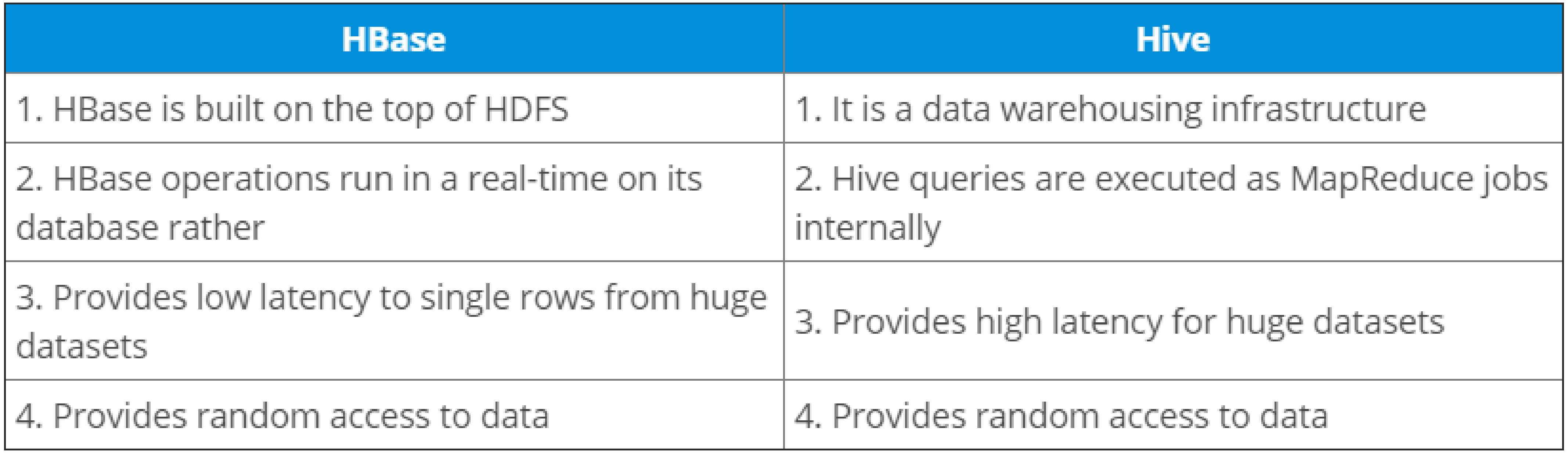


Feature	Hive	Map Reduce
Language	It Supports SQL like query language for interaction and for Data modeling	• It compiles language with two main tasks present in it. One is map task, and another one is a reducer. • We can define these task using Java or Python
Level of abstraction	Higher level of Abstraction on top of HDFS	Lower level of abstraction
Efficiency in Code	Comparatively lesser than Map reduce	Provides High efficiency
Extent of code	Less number of lines code required for execution	More number of lines of codes to be defined
Type of Development work required	Less Development work required	More development work needed

https://www.guru99.com/hive-tutorials.html

Hive vs HBase

Differences between Hive and HBase



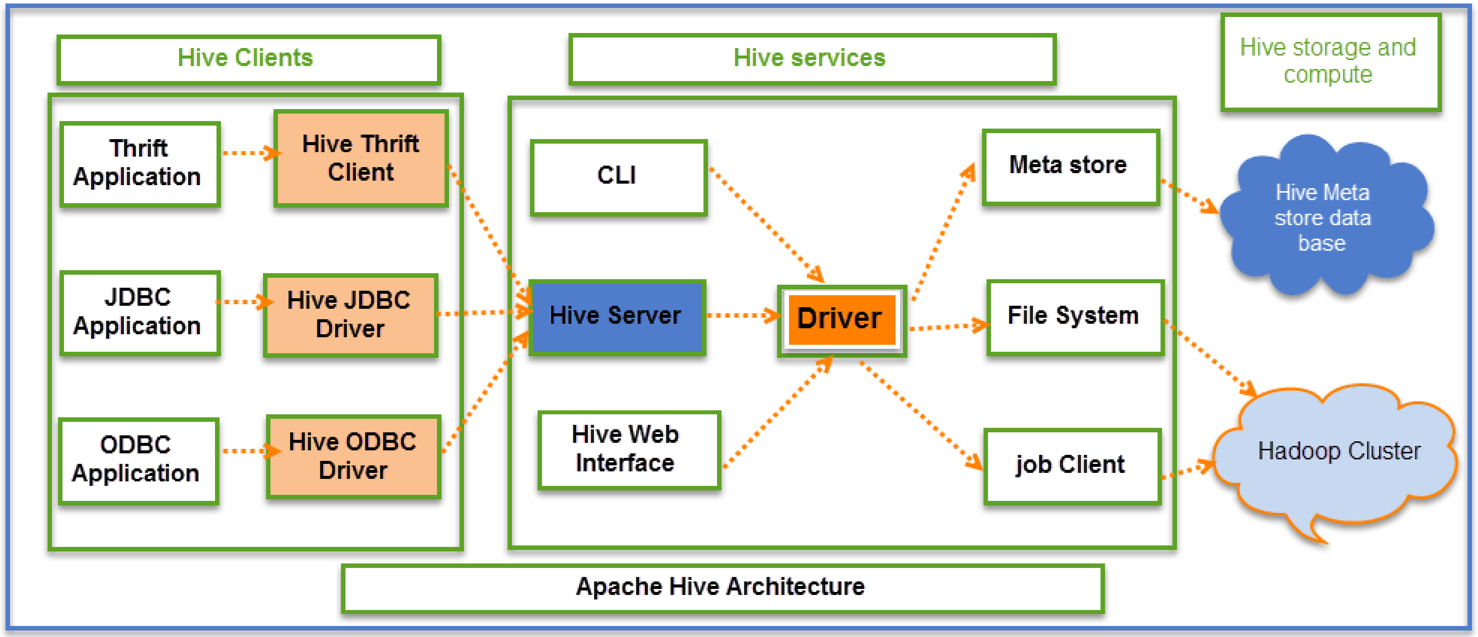
HBase	Hive
1. HBase is built on the top of HDFS	1. It is a data warehousing infrastructure
2. HBase operations run in a real-time on its database rather	2. Hive queries are executed as MapReduce jobs internally
3. Provides low latency to single rows from huge datasets	3. Provides high latency for huge datasets
4. Provides random access to data	4. Provides random access to data

https://www.edureka.co/blog/interview-questions/hive-interview-questions/

Hive Architecture and its Components

Hive Clients:

- They include Thrift application to execute easy Hive commands which are available for python, ruby, C++, and drivers.
- These client application benefits for executing queries on the Hive.
- Hive has three types of client categorization: Thrift clients, JDBC and ODBC clients.

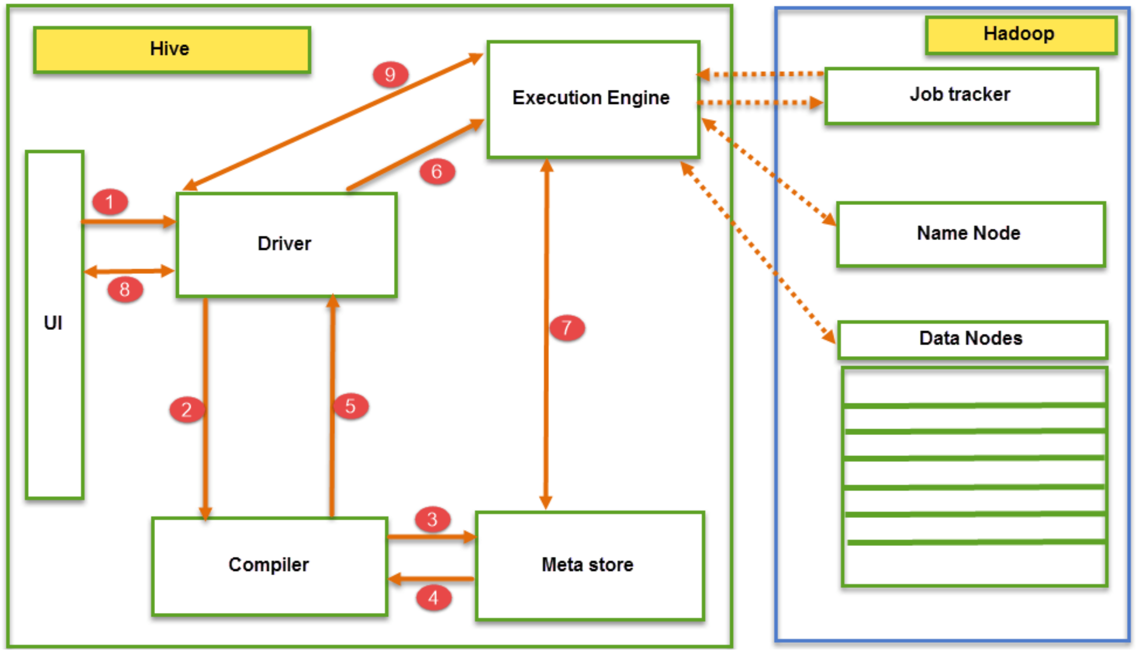


https://www.guru99.com/introduction-hive.html

Hive Architecture and its Components

Hive Services:

- **Metastore:** schema, location in HDFS
- **Driver**: acts like a controller which receives the HiveQL statements. session handles, fetch, execute
- **Compiler:** performs the compilation of the HiveQL query.
- **Optimizer –** performs various transformations on the execution plan to provide optimized **Directed Acyclic Graph (DAG)**.
- **Execution engine:** Once compilation and optimization done, it executes the tasks.
- **CLI, UI, and Thrift Server –** CLI (command-line interface) provides a user interface for an external user to interact with Hive.



https://www.educba.com/hive-architecture/

Hive Architecture and its Components

- Hive data model is structured into partitions, buckets, tables.
- All these can be filtered, have partition keys and to evaluate the query.
- Hive query works on the Hadoop framework, not on the traditional database.
- Hive server is an interface between a remote client queries to the Hive.
- The execution engine is completely embedded in a Hive server.
- Hive application in machine learning, business intelligence in the detection process.

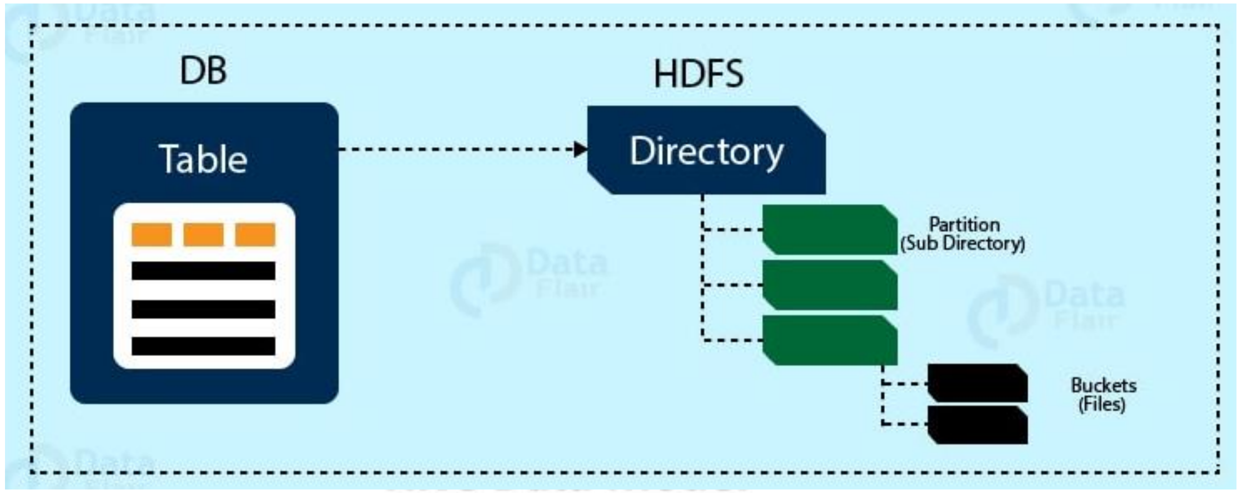
https://www.educba.com/hive-architecture/

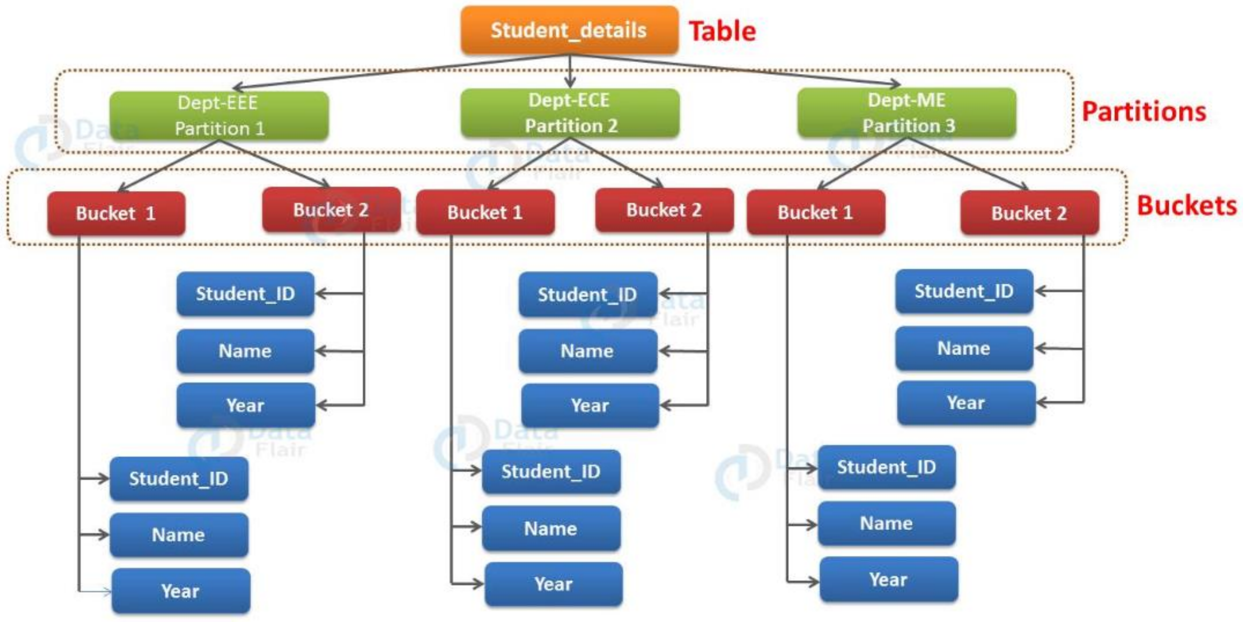
Hive Data Model

- Data in Apache Hive can be categorized into:
  - Tables
    - Typed columns (int, float, string, boolean)
    - Also, list: map (for JSON-like data)
    - *filter, project, join* and *union* operations
  - Partitions
    - For example, range-partition tables by date
  - Buckets
    - Hash partitions within ranges

    (useful for sampling, join optimization)

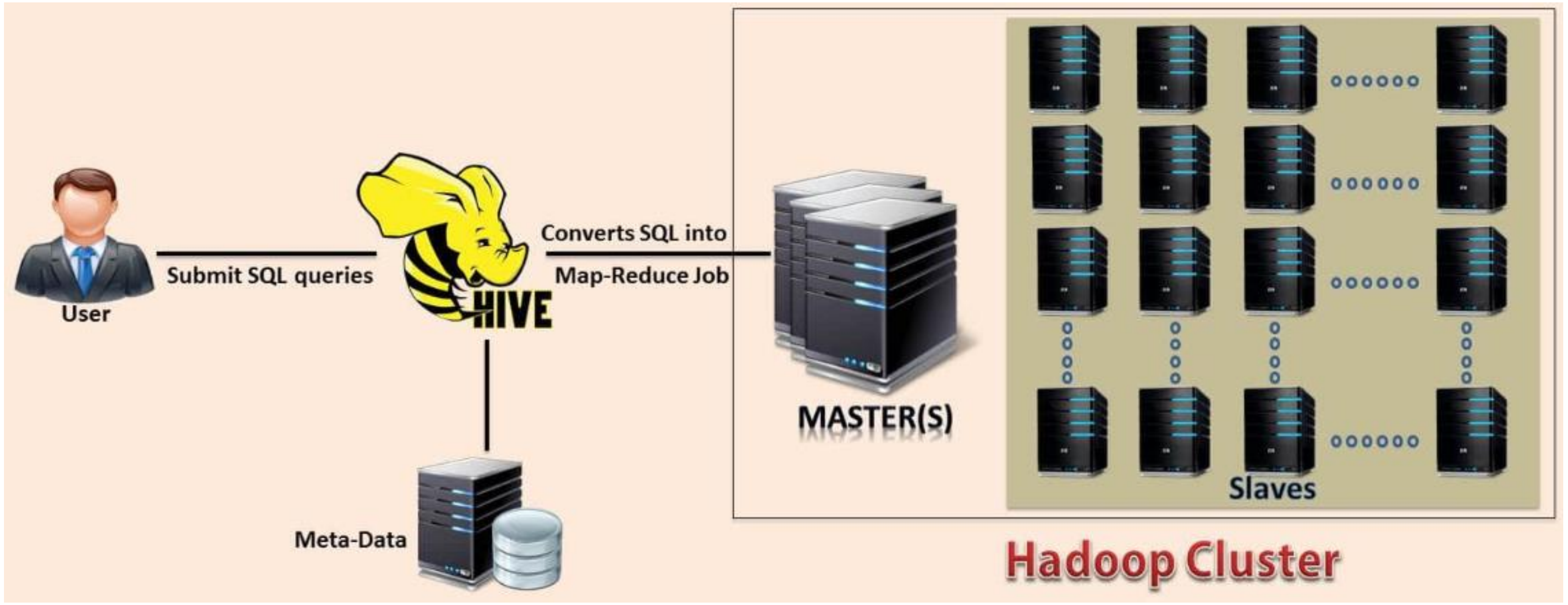
- The Hive table is stored in an HDFS directory /user/hive/warehouse.





https://data-flair.training/blogs/hive-data-model/

Hive and Connection to Hadoop Cluster



https://data-flair.training/blogs/apache-hive-tutorial/

Features of Apache Hive

- Hive provides data summarization, query, and analysis in much easier manner.
- Hive supports external tables; make it possible to process data without actually storing in HDFS.
- It supports partitioning of data at the level of tables to improve performance.
- Hive has a rule-based optimizer for optimizing logical plans.
- It is scalable, familiar, and extensible.
- We can easily process structured data in Hadoop using Hive.
- Querying in Hive is very simple as it is similar to SQL.
- We can also run Ad-hoc queries for the data analysis using Hive.
- We can access files stored in HDFS using Hive or in other data storage systems such as HBase.
- Hive supports more client applications including Java, PHP, Python, C++, Ruby.

https://data-flair.training/blogs/apache-hive-tutorial/

Limitation of Hive

- Hive not designed for Online transaction processing (OLTP), only for Online Analytical Processing.
- Hive supports overwriting or apprehending data, but not updates and deletes.
- In Hive, sub queries are not supported.
- Apache Hive does not offer real-time queries and row level updates.
- Hive also provides acceptable latency for interactive data browsing.
- It is not good for online transaction processing.
- Latency for Apache Hive queries is generally very high.

Hive Metastore

- Metastore in Hive stores the meta data information using RDBMS and an open source ORM (Object Relational Model) layer called Data Nucleus which converts the object representation into relational schema and vice versa.
- Hive stores metadata information in the metastore using RDBMS instead of HDFS. The reason for choosing RDBMS is to achieve low latency as HDFS read/write operations are time consuming processes.
- Metastore is used to hold all the information about the tables and partitions that are in the warehouse.
- Stores Table/Partition properties:
  - Table schema and SerDe library
  - Table Location on HDFS
  - Logical Partitioning keys and types
  - Other information

Hive Metastore

- Provides client access to this information by using metastore service API.
- Database: namespace containing a set of tables.
- By default, the metastore run in the same process as the Hive service; DerBy is default Metastore.
- Metastore can be stored in MySQL and many other relational databases.
- Metadata can be stored as text files or even in a SQL backend.
- Two types of metastore:
  - Local Metastore: the metastore service runs in the same JVM in which the Hive service is running and connects to a database running in a separate JVM, either on the same machine or on a remote machine.
  - Remote Metastore: the metastore service runs on its own separate JVM and not in the Hive service JVM. Other processes communicate with the metastore server using Thrift Network APIs. You can have one or more metastore servers in this case to provide more availability.

https://www.edureka.co/blog/interview-questions/hive-interview-questions/

Hive Metastore

- Stores metadata for Hive tables (e.g. schema and location) and partitions in a relational database
- Metastore is used to hold all the information about the tables and partitions that are in the warehouse.
- Stores Table/Partition properties:
  - Table schema and SerDe library
  - Table Location on HDFS
  - Logical Partitioning keys and types
  - Other information
- Provides client access to this information by using metastore service API
- Database: namespace containing a set of tables
- By default, the metastore run in the same process as the Hive service; DerBy is default Metastore.
- Metastore can be stored in MySQL and many other relational databases
- Metadata can be stored as text files or even in a SQL backend

Physical Layout

- Warehouse directory in HDFS
  - E.g., /user/hive/warehouse
- Tables stored in subdirectories of warehouse
  - Partitions form subdirectories of tables
- Actual data stored in flat files
  - Control char-delimited text, or SequenceFiles
  - With custom SerDe, can use arbitrary format

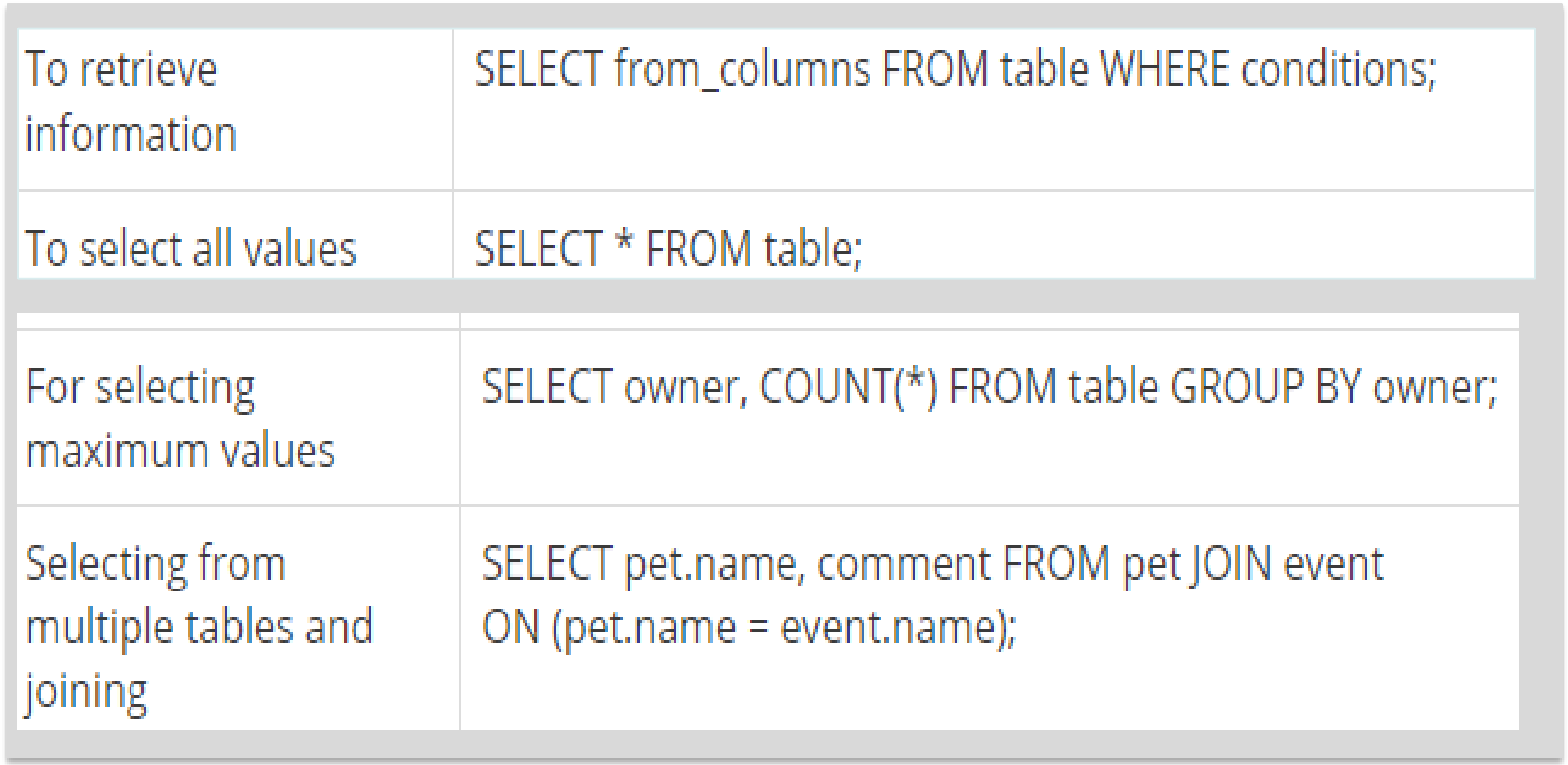
Hive CLI

- DDL:
  - create table/drop table/rename table
  - alter table add column
- Browsing:
  - show tables
  - describe table
  - cat table
- Loading Data
- Queries

What is HiveQL?

- Hive: data warehousing application on top of HDFS in Hadoop
  - Provides a SQL like dialect to interact with data
  - Query language is HiveQL, queries similar to SQL (an SQL-like engine that runs MapReduce jobs)
  - Tables stored on HDFS as flat files
  - by Facebook, open source

Hive syntax using HQL:



To retrieve information	SELECT from_columns FROM table WHERE conditions;
To select all values	SELECT * FROM table;
For selecting maximum values	SELECT owner, COUNT(*) FROM table GROUP BY owner;
Selecting from multiple tables and joining	SELECT pet.name, comment FROM pet JOIN event ON (pet.name = event.name);

Hive Language: Example

- Hive looks similar to an SQL database
- Relational **join** on two tables:
  - Table of word counts from Shakespeare collection
  - Table of word counts from the bible

Example:

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word)
WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```

Hive: Behind the Scenes

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
        JOIN bible k ON (s.word = k.word)
        WHERE s.freq >= 1 AND k.freq >= 1
        ORDER BY s.freq DESC LIMIT 10;
```

(Abstract Syntax Tree)

```
(TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF shakespeare s) (TOK_TABREF bible k) (= (. (TOK_TABLE_OR_COL s)
word) (. (TOK_TABLE_OR_COL k) word)))) (TOK_INSERT (TOK_DESTINATION (TOK_DIR TOK_TMP_FILE)) (TOK_SELECT
(TOK_SELEXPR (. (TOK_TABLE_OR_COL s) word)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL s) freq)) (TOK_SELEXPR (.
(TOK_TABLE_OR_COL k) freq))) (TOK_WHERE (AND (>= (. (TOK_TABLE_OR_COL s) freq) 1) (>= (. (TOK_TABLE_OR_COL k)
freq) 1))) (TOK_ORDERBY (TOK_TABSORTCOLNAMEDESC (. (TOK_TABLE_OR_COL s) freq))) (TOK_LIMIT 10)))
```

(one or more of MapReduce jobs)

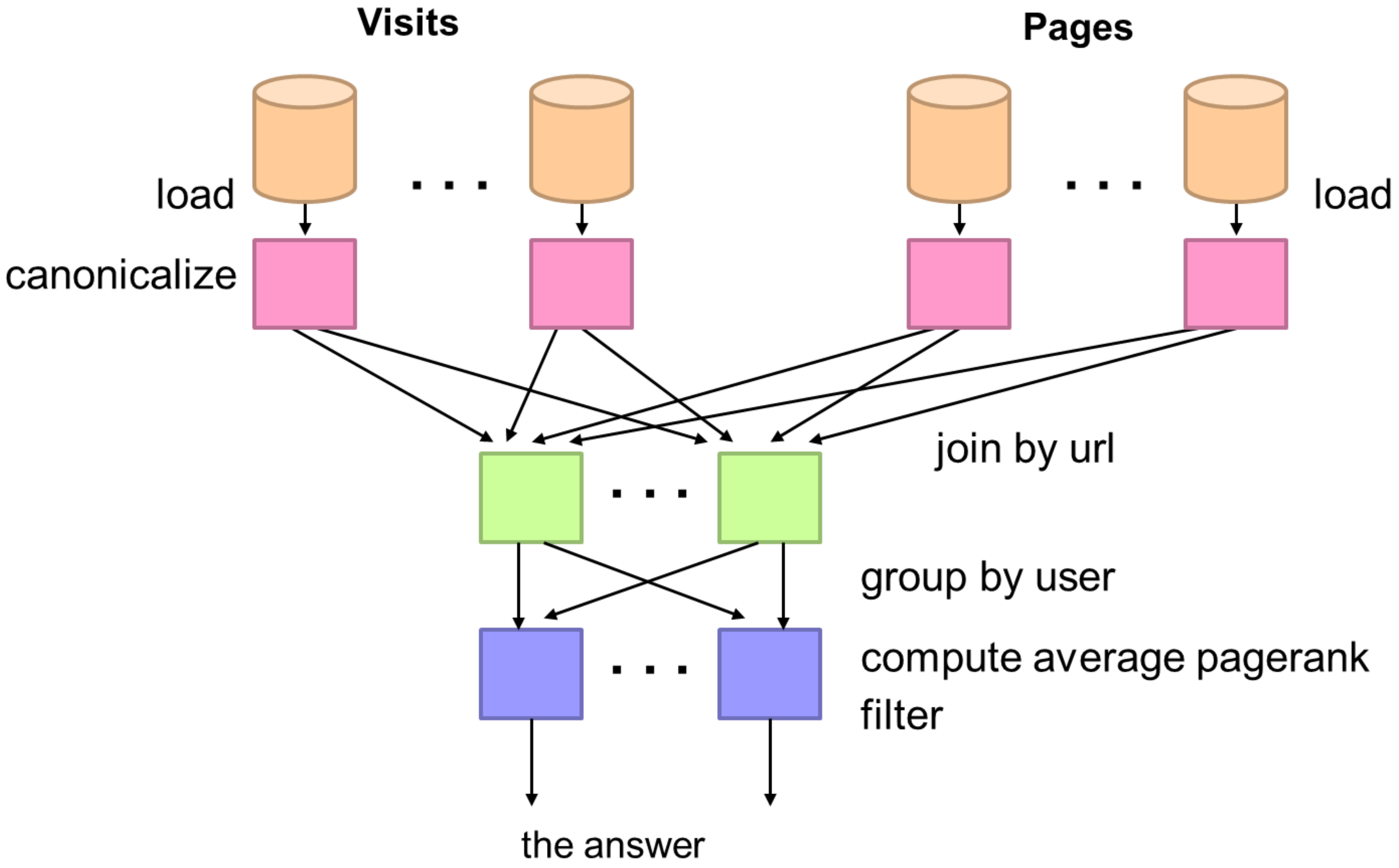
Hive: Behind the Scenes

```
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-2 depends on stages: Stage-1
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Alias -> Map Operator Tree:
        s
          TableScan
            alias: s
            Filter Operator
              predicate:
                  expr: (freq >= 1)
                  type: boolean
              Reduce Output Operator
                key expressions:
                      expr: word
                      type: string
                sort order: +
                Map-reduce partition columns:
                      expr: word
                      type: string
                tag: 0
                value expressions:
                      expr: freq
                      type: int
                      expr: word
                      type: string
        k
          TableScan
            alias: k
            Filter Operator
              predicate:
                  expr: (freq >= 1)
                  type: boolean
              Reduce Output Operator
                key expressions:
                      expr: word
                      type: string
                sort order: +
                Map-reduce partition columns:
                      expr: word
                      type: string
                tag: 1
                value expressions:
                      expr: freq
                      type: int
```

```
Reduce Operator Tree:
       Join Operator
         condition map:
              Inner Join 0 to 1
         condition expressions:
           0 {VALUE._col0} {VALUE._col1}
           1 {VALUE._col0}
         outputColumnNames: _col0, _col1, _col2
         Filter Operator
           predicate:
               expr: ((_col0 >= 1) and (_col2 >= 1))
               type: boolean
           Select Operator
             expressions:
                   expr: _col1
                   type: string
                   expr: _col0
                   type: int
                   expr: _col2
                   type: int
             outputColumnNames: _col0, _col1, _col2
             File Output Operator
               compressed: false
               GlobalTableId: 0
               table:
                   input format: org.apache.hadoop.mapred.SequenceFileInputFormat
                   output format: org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat
```

```
Stage: Stage-2
   Map Reduce
     Alias -> Map Operator Tree:
       hdfs://localhost:8022/tmp/hive-training/364214370/10002
           Reduce Output Operator
             key expressions:
                   expr: _col1
                   type: int
             sort order: -
             tag: -1
             value expressions:
                   expr: _col0
                   type: string
                   expr: _col1
                   type: int
                   expr: _col2
                   type: int
     Reduce Operator Tree:
       Extract
         Limit
           File Output Operator
             compressed: false
             GlobalTableId: 0
             table:
                 input format: org.apache.hadoop.mapred.TextInputFormat
                 output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat


Stage: Stage-0
   Fetch Operator
     limit: 10
```

System-Level Dataflow



MapReduce Code

```java
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.a pache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobC   ontrol;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.sub  string(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1  " + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                 OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(  firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so w   e know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.to  String();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                 }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputColle  ctor<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', first   Comma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text ke y,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            wh ile (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        i mplements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter)   throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public  void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count  < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.se tJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/ user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.s etJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.add  InputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(  MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMap  per.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.se  tOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRE  xample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFi  leOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
         FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputF  ormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
         FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top   100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

Integration - Hive

- Reasons to use Hive on HBase:
  - A lot of data sitting in HBase due to its usage in a real-time environment, but never used for analysis
  - Give access to data in HBase usually only queried through MapReduce to people that don’t code (business analysts)
  - When needing a more flexible storage solution, so that rows can be updated live by either a Hive job or an application and can be seen immediately to the other

- Reasons not to do it:
  - Run SQL queries on HBase to answer live user requests (it’s still a MR job)
  - Hoping to see interoperability with other SQL analytics systems

Use Cases

- Front-end engineers
  - They need some statistics regarding their latest product
- Research engineers
  - Ad-hoc queries on user data to validate some assumptions
  - Generating statistics about recommendation quality
- Business analysts
  - Statistics on growth and activity
  - Effectiveness of advertiser campaigns
  - Users’ behavior VS past activities to determine, for example, why certain groups react better to email communications
  - Ad-hoc queries on stumbling behaviors of slices of the user base

Use Cases

- Using a simple table in HBase:

```
CREATE EXTERNAL TABLE blocked_users(
 userid INT,
 blockee INT,
 blocker INT,
 created BIGINT)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler’
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
 ":key,f:blockee,f:blocker,f:created")
TBLPROPERTIES("hbase.table.name" = "m2h_repl-userdb.stumble.blocked_users");
```

HBase is a special case here, it has a unique row key map with :key
Not all the columns in the table need to be mapped

Use Cases

- Using a complicated table in HBase:

```
CREATE EXTERNAL TABLE ratings_hbase(
 userid INT,
 created BIGINT,
 urlid INT,
 rating INT,
 topic INT,
 modified BIGINT)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler’
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
 ":key#b@0,:key#b@1,:key#b@2,default:rating#b,default:topic#b,default:modified#b")
TBLPROPERTIES("hbase.table.name" = "ratings_by_userid");
```

#b means binary, @ means position in composite key (SU-specific hack)

Dealing with Structured Data

- Type system
  - Primitive types
  - Recursively build up using Composition/Maps/Lists
- Generic (De)Serialization Interface (SerDe)
  - To recursively list schema
  - To recursively access fields within a row object
- Serialization families implement interface
  - Thrift DDL based SerDe
  - Delimited text based SerDe
  - You can write your own SerDe
- Schema Evolution

Hive Query Language

- Philosophy
  - SQL like constructs + Hadoop Streaming

- Query Operators in initial version
  - Projections
  - Equijoins and Cogroups
  - Group by
  - Sampling

- Output of these operators can be:
  - passed to Streaming mappers/reducers
  - can be stored in another Hive Table
  - can be output to HDFS files
  - can be output to local files

Hive Query Language

- Package these capabilities into a more formal SQL like query language in next version
- Introduce other important constructs:
  - Ability to stream data thru custom mappers/reducers
  - Multi table inserts
  - Multiple group bys
  - SQL like column expressions and some XPath like expressions
  - Etc..

Joins

- Joins

```
FROM page_view pv JOIN user u ON (pv.userid = u.id)
INSERT INTO TABLE pv_users
SELECT pv.*, u.gender, u.age
WHERE pv.date = 2008-03-03;
```

- Outer Joins

```
FROM page_view pv FULL OUTER JOIN user u ON (pv.userid = u.id) INSERT INTO TABLE
pv_users
SELECT pv.*, u.gender, u.age
WHERE pv.date = 2008-03-03;
```

Aggregations and Multi-Table Inserts

```
FROM pv_users
INSERT INTO TABLE pv_gender_uu
    SELECT pv_users.gender, count(DISTINCT pv_users.userid)
    GROUP BY(pv_users.gender)
INSERT INTO TABLE pv_ip_uu
    SELECT pv_users.ip, count(DISTINCT pv_users.id)
    GROUP BY(pv_users.ip);
```

Running Custom Map/Reduce Scripts

```
FROM (
    FROM pv_users
    SELECT TRANSFORM(pv_users.userid, pv_users.date) USING 'map_script'
    AS(dt, uid)
    CLUSTER BY(dt)) map
INSERT INTO TABLE pv_users_reduced
    SELECT TRANSFORM(map.dt, map.uid) USING 'reduce_script' AS (date, count);
```

Inserts into Files, Tables and Local Files

```
FROM pv_users
INSERT INTO TABLE pv_gender_sum
    SELECT pv_users.gender, count_distinct(pv_users.userid)
    GROUP BY(pv_users.gender)
INSERT INTO DIRECTORY ‘/user/facebook/tmp/pv_age_sum.dir’
    SELECT pv_users.age, count_distinct(pv_users.userid)
    GROUP BY(pv_users.age)
INSERT INTO LOCAL DIRECTORY ‘/home/me/pv_age_sum.dir’
    FIELDS TERMINATED BY ‘,’ LINES TERMINATED BY \013
    SELECT pv_users.age, count_distinct(pv_users.userid)
    GROUP BY(pv_users.age);
```

Pig

- Pig is a high level scripting language that is used with Apache Hadoop.
- It enables writing complex data transformations without knowing Java.
- Pig's simple SQL-like scripting language is called Pig Latin.
- Apache Pig follows ETL (Extract Transform Load) process.
- Apache Pig automatically optimizes the tasks before execution, and handles all kinds of data.
- Pig allows programmers to write custom functions which is unavailable in Pig.
- User Defined Functions can be written in different language like Java, Python, Ruby, etc. and embed them in Pig script.
- Pig Latin provides various built-in operators like join, sort, filter, etc. to read, write, and process large data sets.

Pig – Advantages vs limitations

Benefits of Apache Pig	Limitations
• Less development time • Easy to learn • Procedural language • Dataflow • Easy to control execution • UDFs • Lazy evaluation • Usage of Hadoop features • Effective for unstructured • Base Pipeline	• Errors of Pig • Not mature • Support • Minor one • Implicit data schema • Delay in execution

https://data-flair.training/blogs/pig-advantages-and-disadvantages/

Pig Latin Script

Command example:

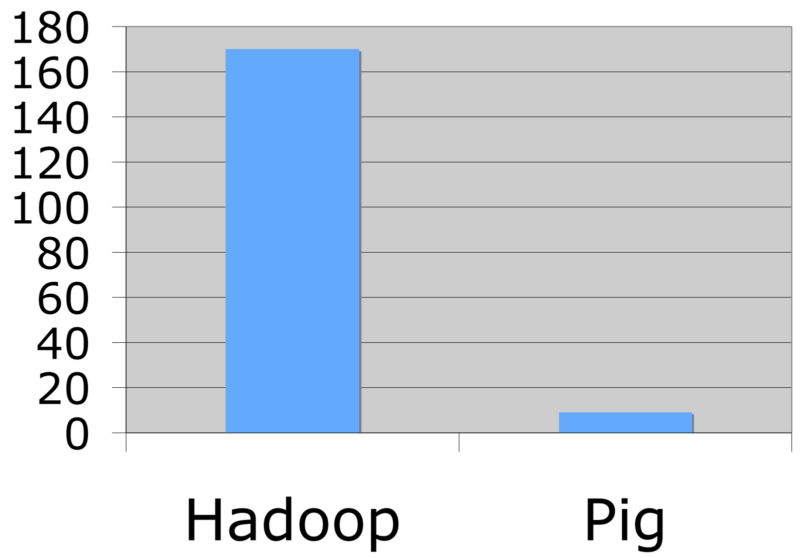
```
Visits = load    ‘/data/visits’ as (user, url, time);
Visits = foreach Visits generate user, Canonicalize(url), time;

Pages = load    ‘/data/pages’ as (url, pagerank);

VP = join    Visits by url, Pages by url;
UserVisits = group   VP by user;
UserPageranks = foreach UserVisits generate user,
AVG(VP.pagerank) as avgpr;
GoodUsers = filter  UserPageranks by avgpr > ‘0.5’;

store   GoodUsers into '/data/good_users';
```

More details on Pig:

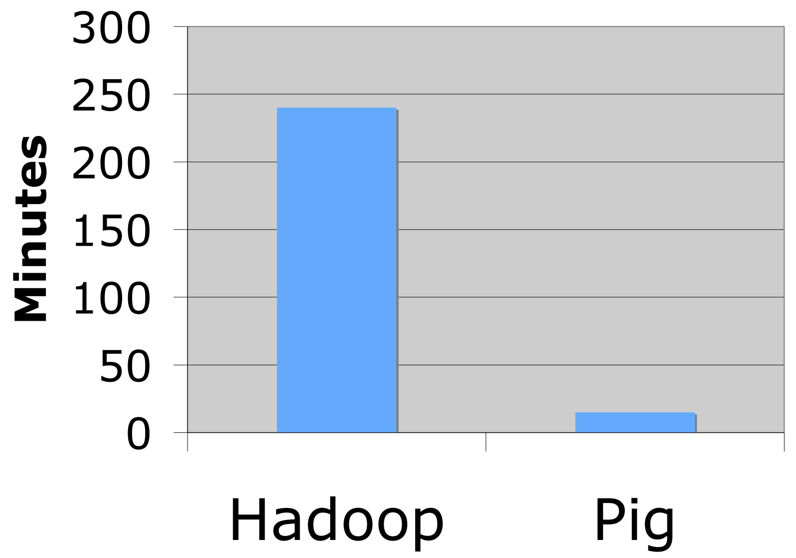
https://www.slideshare.net/martyhall/hadoop-tutorial-pig-part-1-introduction-to-apache-pig

Java vs. Pig Latin

1/20 the lines of code



1/16 the development time

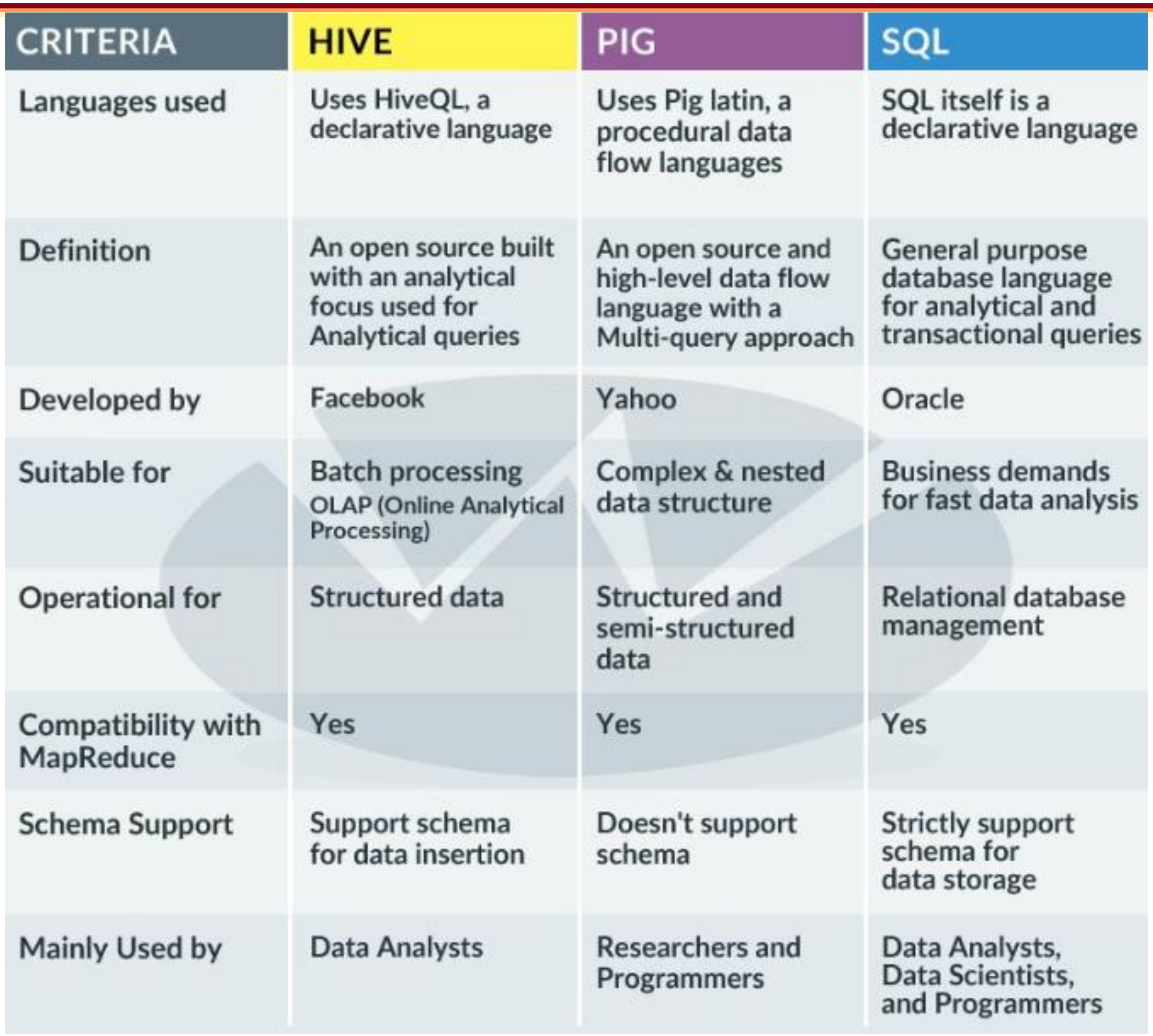


Performance on par with raw Hadoop!

Pig takes care of…

- Schema and type checking
- Translating into efficient physical dataflow
  - (i.e., sequence of one or more MapReduce jobs)
- Exploiting data reduction opportunities
  - (e.g., early partial aggregation via a combiner)
- Executing the system-level dataflow
  - (i.e., running the MapReduce jobs)
- Tracking progress, errors, etc.

Technical Differences Between Hive vs Pig vs SQL



CRITERIA	HIVE	PIG	SQL
Languages used	Uses HiveQL, a declarative language	Uses Pig latin, a procedural data flow languages	SQL itself is a declarative language
Definition	An open source built with an analytical focus used for Analytical queries	An open source and high-level data flow language with a Multi-query approach	General purpose database language for analytical and transactional queries
Developed by	Facebook	Yahoo	Oracle
Suitable for	Batch processing OLAP (Online Analytical Processing)	Complex & nested data structure	Business demands for fast data analysis
Operational for	Structured data	Structured and semi-structured data	Relational database management
Compatibility with MapReduce	Yes	Yes	Yes
Schema Support	Support schema for data insertion	Doesn't support schema	Strictly support schema for data storage
Mainly Used by	Data Analysts	Researchers and Programmers	Data Analysts, Data Scientists, and Programmers

https://www.whizlabs.com/blog/hive-vs-pig-vs-sql/

The End