Are you ready **for the future?**

High Performance Solutions, Backend Development and Integration Services, Embedded Software Development, Big Data (Visualization, Architecture, Science), Business Intelligence & Analytics

Globant
we are ready

BIG DATA &
HIGH PERFORMANCE

AUG 2014

The following presentation is based on
the Apache Hive Documentation ,
Programming Hive book, Hadoop the Definitive Guide
and the experience of some developers from
the Big Data Studio
for internal training purposes.

Hive

Hive - What it is

What it is

- Structured Data Warehouse over Hadoop
- SQL -> Map Reduce
- HDFS Storage
- Metadata in RDBMS
- Aggregation functions

What it is NOT

- SchemalessDB
- TransactionalDB
- Real Time platform

Hive - What it is

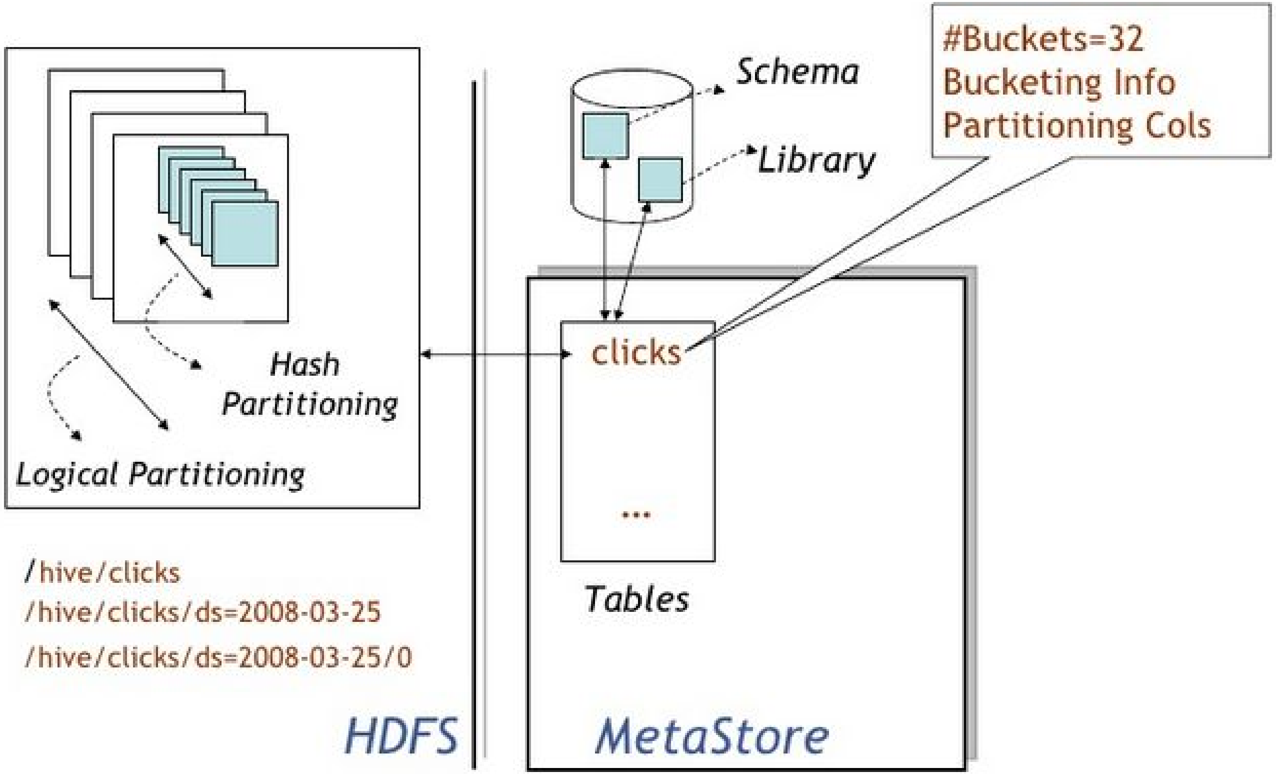
What it is NOT

Hive is **not** designed for online transaction processing (OLTP) workloads.

It is best used for traditional data warehousing tasks.

Hive - Architecture

Data Model



Hive - Shell Client (hive cli)

1) Interactive:

```
hive
```

2) Non Interactive:

```
hive -S -e "select * from ventas ;"


-S   -v   -f   -e  --hivevar   --hiveconf
```

Hive - Data Types: Basic

TINYINT: 1 byte signed integer.

SMALLINT: 2 byte signed integer.

INT: 4 byte signed integer.

BIGINT: 8 byte signed integer.

BOOLEAN: Boolean true or false.

FLOAT: Single precision floating point.

DOUBLE: Double precision floating point.

STRING: Sequence of characters.

TIMESTAMP: Integer, float, or string. Compatible con java.sql.Timestamp

BINARY: Array of bytes.

Hive - Data Types: Complex

- ➢ ARRAY array('Juan', 'Luis')

Ordered sequences of the same type that are indexable using zero-based integers. For example, if a column name is of type ARRAY of strings with the value ['John', 'Doe'], then

the second element can be referenced using name[1].

- ➢ MAP map('Nombre': 'Juan, 'Apellido': 'Perez')

A collection of key-value tuples, where the fields are accessed using array notation (e.g., ['key']).

For example, if a column name is of type MAP with key→value pairs

'first'→'John' and 'last'→'Doe', then the last

name can be referenced using name['last'].

- ➢ STRUCT: ('Nombre':'Juan', 'Edad':40, 'Sexo':'M')

Analogous to a C struct or an “object.” Fields can be accessed using the “dot” notation.

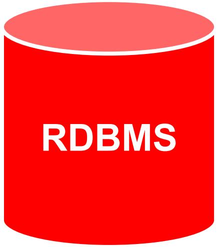
For example, if a column name is of:

type STRUCT:name {first STRING; last STRING}, then

the first name field can be referenced using name.first.

Hive - Schema On Read

Schema on WRITE



1 Create schema

2 Load Data

3 Read Data

✗ Stuck with one-size-fits all schema

✗ Complex with multiple-datasets

✗ Bad for Unknown datasets

Schema on READ

1 Load Data as-is

2 Create Schema

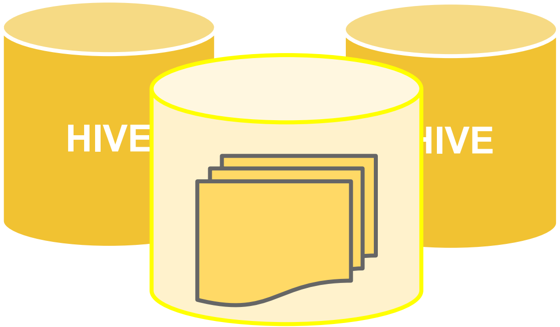
3 Read it

⇨ If the schema doesn’t match, it fills the field with null values

✚ Useful to work with multiple datasets

✚ Good for know datasets

Internal Tables



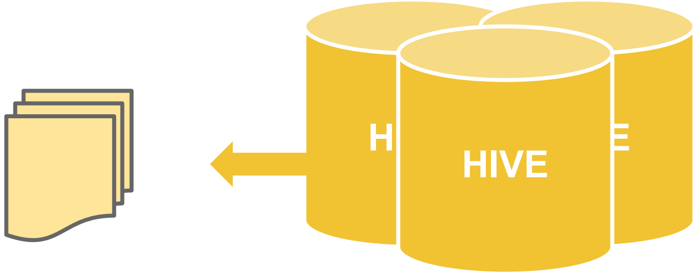
- Hive controls the lifecycle of their data.
- Hive stores the data for these tables in a subdirectory under the directory defined by hive.metastore.warehouse.dir
- Less convenient for shared data between different tools

!

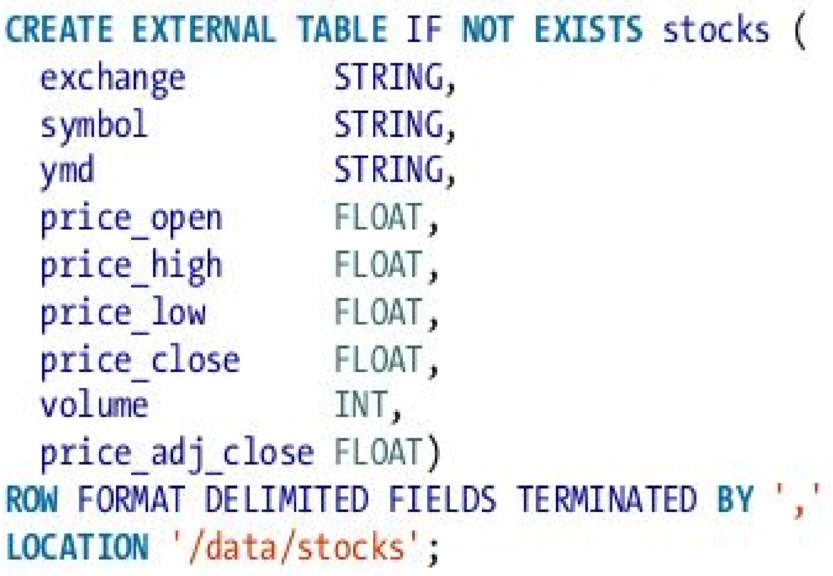
When we **DROP** an internal table

Hive deletes the data inside the table

External Tables

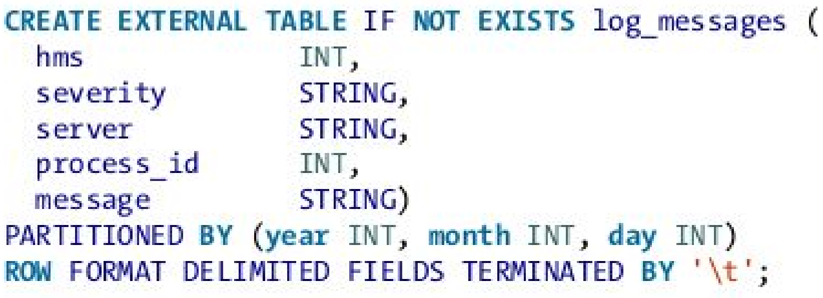


- Hive does not assume it owns the data.
- **Dropping the table does not delete the data**,
- (although the metadata for the table will be deleted)



```
CREATE EXTERNAL TABLE IF NOT EXISTS stocks (
  exchange        STRING,
  symbol          STRING,
  ymd             STRING,
  price_open      FLOAT,
  price_high      FLOAT,
  price_low       FLOAT,
  price_close     FLOAT,
  volume          INT,
  price_adj_close FLOAT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/data/stocks';
```

Partitioned Tables



```
CREATE EXTERNAL TABLE IF NOT EXISTS log_messages (
  hms             INT,
  severity        STRING,
  server          STRING,
  process_id      INT,
  message         STRING)
PARTITIONED BY (year INT, month INT, day INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

hdfs:://master/user/hive/log_messages

./employees/year=2012/month=October/day=12
./employees/year=2012/month=November/day=12

Users of the table don’t need to care if these “columns”are partitions or not, except when they want to **optimize query performance**

You can declare a folder as a partition.

If we add new data to a folder declared as partition, that data will appear available for the table after the insertion

A query without a WHERE clause will query all the data in all the partitions.
If you add the WHERE clause, it will read only the partition (which is translated as accessing a folder) that match with that condition.

However, the partition you create makes a pseudocolumn on which you can query

Bucketed Tables

```
CREATE TABLE bucketed_users (
  id INT,
  name STRING)
CLUSTERED BY (id) INTO 4 BUCKETS;

set hive.enforce.bucketing= true;
```

```
hdsfs::/master/user/hive/bucketed_users

  ./00000_0
  ./00000_1
  ./00000_2
  ./00000_3
```

Data is stored in as many files as buckets inside the partition/table folder.

User unaware if table is bucketed or not

⊕ Bucketed tables are fantastic in that they allow much more efficient sampling

⊕ Allows **efficient queries,** e.g., “map-side join”

A Clustered Field value is assigned a bucket, the Record is put on the file of that bucket (similar to HashPartitioner.class in mapreduce)

```
set hive.enforce.bucketing= true;
```

sets same number of reducers as buckets during inserts to table.

Bucketed Tables

```
CREATE TABLE page_view(viewTime INT, userid
BIGINT,
     page_url STRING, referrer_url STRING,
     ip STRING COMMENT 'IP Address of the
User')
 COMMENT 'This is the page view table'
 PARTITIONED BY(dt STRING, country STRING)
 CLUSTERED BY(userid) SORTED BY(viewTime)
INTO 32 BUCKETS
 ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\001'
   COLLECTION ITEMS TERMINATED BY '\002'
   MAP KEYS TERMINATED BY '\003'
 STORED AS SEQUENCEFILE;
```

The page_view table is bucketed (clustered by) *userid* and within each bucket the data is sorted in increasing order of viewTime. Such an organization allows the user to **do efficient sampling** on the clustered column - in this case userid

The CLUSTERED BY and SORTED BY creation commands do not affect how data is inserted into a table – only **how it is read**.

Users must be careful to insert data correctly by specifying the number of reducers to be equal to the number of buckets, and using CLUSTER BY and SORT BY commands in their query.

Temporary Tables & Views

TEMPORARY TABLES

- Will only be visible to the current session
- No support for creation of indexes
- Partition columns are not supported.

VIEWS

- A purely logical object with no associated storage
- A view's schema is frozen at the time the view is created; subsequent changes to underlying tables (e.g. adding a column) will not be reflected in the view's schema.
- Views are read-only

Hive - Reading files...Row way

```
CREATE TABLE usuarios(
    nombre STRING,
    edad INT)
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '\t'
    LINES TERMINATED BY '\n';
```

Hive - Reading file...SerDe way

- ➢ HDFS Files → InputFileFormat → <key, value> → Deserializer → Row object
  - ○ Example: Query data
- ➢ Row object → Serializer → <key, value> → OutputFileFormat → HDFS Files
  - ○ Insert

```
CREATE EXTERNAL TABLE tweets (
   ...
   retweeted_status STRUCT<
      text:STRING,
      user:STRUCT<
         screen_name:STRING,
         name:STRING
      >
   >,
   text STRING,
   ...
  )
ROW FORMAT SERDE
   'com.cloudera.hive.serde.JSONSerDe'
```

Hive - Reading files...Serde way

Using Json SerDe you can save storage

If your schema has a lot of columns and often they are null. It's a good idea to use SerDe. You can create json without the keys (columns) and Serde will show it as null.

Loading Data into a table

```
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)]
```

In this case, load operations are currently **pure copy/move operations** that move datafiles into locations corresponding to Hive tables.

Since Hive 0.14:

```
INSERT INTO TABLE tablename [PARTITION (partcol1[=val1], partcol2[=val2] ...)] VALUES values_row [, values_row ...]
```

Compressed Data Storage :Keeping data compressed in Hive tables has, in some cases, been known to give **better performance** than uncompressed storage; both in terms of **disk usage** and **query performance**.

You can import text files compressed with Gzip or Bzip2 directly into a table stored as TextFile

However, in this case Hadoop will not be able to split your file into chunks/blocks and run multiple maps in parallel.

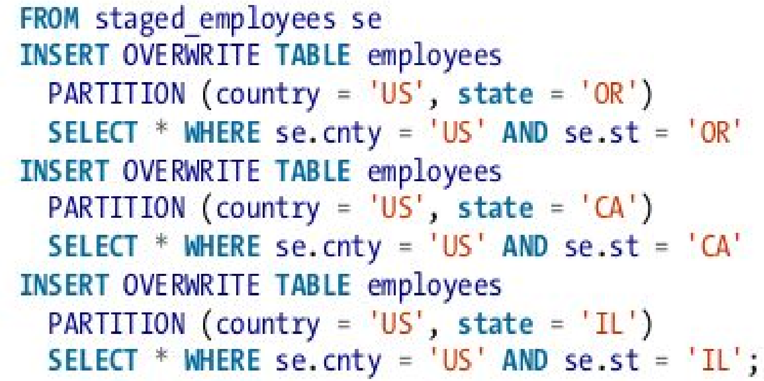
Loading Data into a table with another format

```
CREATE TABLE raw (line STRING)
   ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n';

CREATE TABLE raw_sequence (line STRING)
   STORED AS SEQUENCEFILE;

LOAD DATA LOCAL INPATH '/tmp/weblogs/20090603-access.log.gz' INTO TABLE raw;

SET hive.exec.compress.output=true;
SET io.seqfile.compression.type=BLOCK; -- NONE/RECORD/BLOCK (see below)
INSERT OVERWRITE TABLE raw_sequence SELECT * FROM raw;
```

Inserting Data into a table

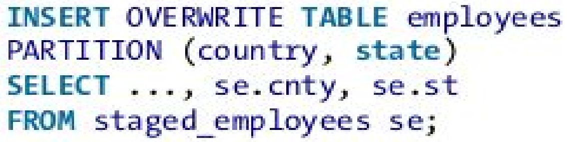
```
//Overwrite data in tablename1
INSERT OVERWRITE TABLE tablename1 [PARTITION
(partcol1=val1, partcol2=val2 ...) [IF NOT
EXISTS]]
select_statement1 FROM from_statement;

// Append the data to tablename1
INSERT INTO TABLE tablename1
[PARTITION (partcol1=val1, partcol2=val2
...)]
select_statement1 FROM from_statement;
```

Static Partition Insert : Define partitions in the insert query.



```
FROM staged_employees se
INSERT OVERWRITE TABLE employees
  PARTITION (country = 'US', state = 'OR')
  SELECT * WHERE se.cnty = 'US' AND se.st = 'OR'
INSERT OVERWRITE TABLE employees
  PARTITION (country = 'US', state = 'CA')
  SELECT * WHERE se.cnty = 'US' AND se.st = 'CA'
INSERT OVERWRITE TABLE employees
  PARTITION (country = 'US', state = 'IL')
  SELECT * WHERE se.cnty = 'US' AND se.st = 'IL';
```

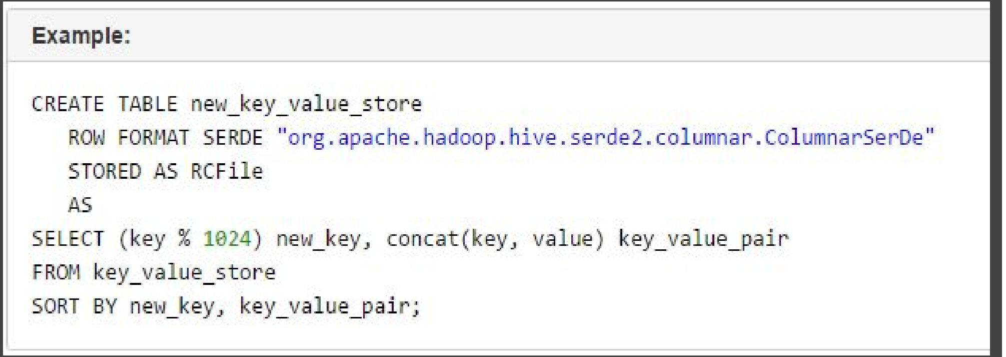
Dynamic Partition Insert : Infer the partitions to create based on query parameters.



```
INSERT OVERWRITE TABLE employees
PARTITION (country, state)
SELECT ..., se.cnty, se.st
FROM staged_employees se;
```

Inserting Data into a table - Create Table as Select (CTAS)

The table created by CTAS is atomic, meaning that the table is not seen by other users until all the query results are populated.



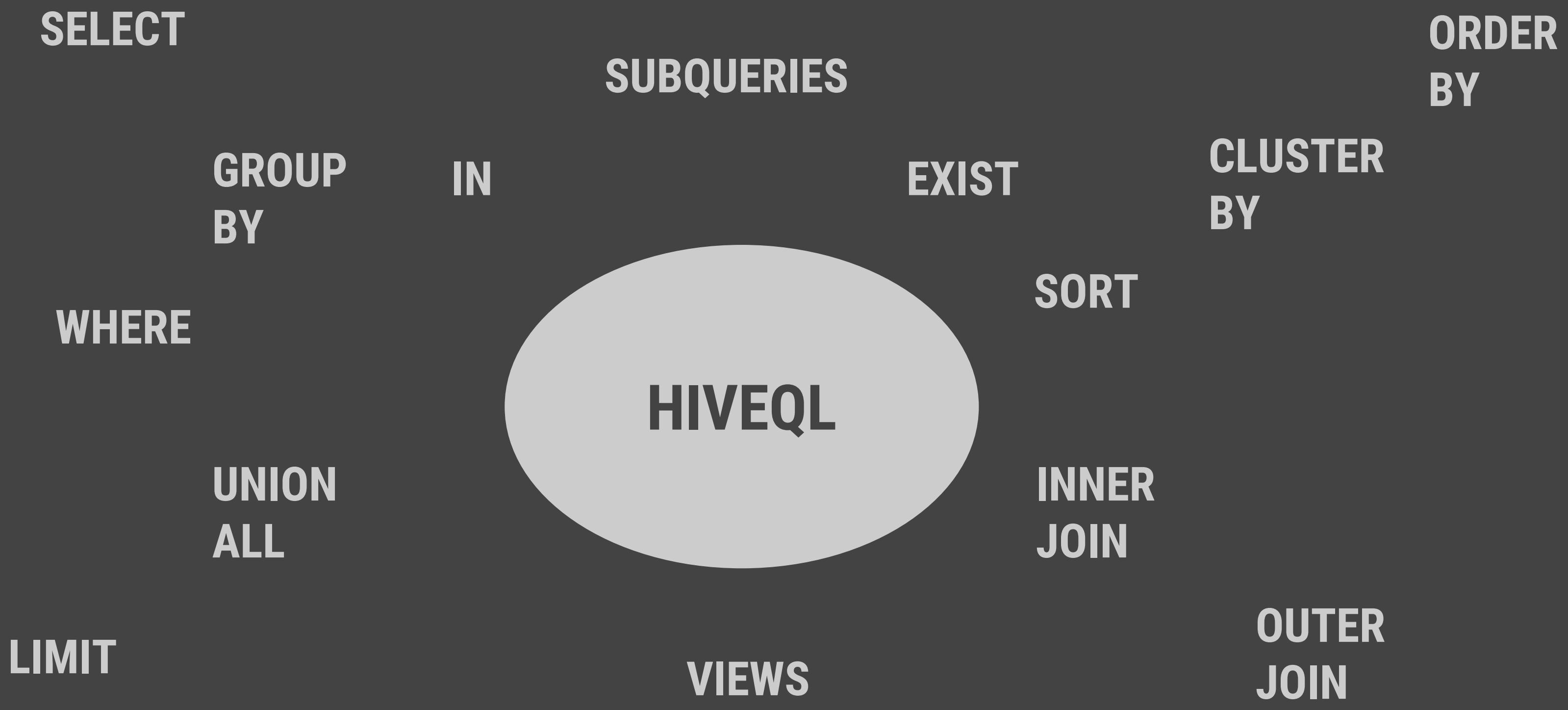
Example:

```
CREATE TABLE new_key_value_store
   ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe"
   STORED AS RCFile
   AS
SELECT (key % 1024) new_key, concat(key, value) key_value_pair
FROM key_value_store
SORT BY new_key, key_value_pair;
```

CTAS has these restrictions:

- The target table cannot be a partitioned table.
- The target table cannot be an external table.
- The target table cannot be a list bucketing table.

HiveQL: Queries - SELECT



HIVEQL

SELECT

SUBQUERIES

ORDER BY

GROUP BY

IN

EXIST

CLUSTER BY

SORT

WHERE

UNION ALL

INNER JOIN

LIMIT

VIEWS

OUTER JOIN

HiveQL: Queries - SELECT

```
> SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[CLUSTER BY col_list  | [DISTRIBUTE BY col_list] [SORT BY col_list]]
[LIMIT number];


> SELECT * FROM log_messages WHERE year = 2014 AND month = ‘December’;


> SELECT name FROM employees WHERE surname LIKE ‘%Mess.’;
```

HiveQL: Queries - JOINS I

USE **EXPLAIN** TO CHECK PERFORMANCE

```
> SELECT a.* FROM a JOIN b ON (a.id = b.id AND a.department = b.department);
```

> NOT ALLOWED(NON-Equality): ~~SELECT a.* FROM a JOIN b ON (a.id <> b.id);~~

~~#~~ Hive converts joins over multiple tables into a single map/reduce job if for every table the same column is used in the join clause. if not, it is converted into multiple map/reduce jobs

```
> SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key1) #1 M-R
```

#OUTER JOINS: Outer join allow you to find nonmatches in the tables being joined.

```
> SELECT sales.* FROM sales RIGHT OUTER JOIN employee ON sales.employeeid = employee.id
```

#SEMI JOINS

```
> SELECT * FROM singers WHERE singers.id IN (SELECT sales.singerId from sales)
```

HiveQL: Queries - JOINS II

USE **EXPLAIN** TO CHECK PERFORMANCE

#MAP JOINS: If one table fits in memory, Hive can load it into memory in the map phase. It does not need a reducer.

```
> SELECT /*+ MAPJOIN(b) */ a.key, a.value FROM a JOIN b ON a.key = b.key;
> SELECT /*+ MAPJOIN(c) */ * FROM orders o JOIN cities c ON (o.city_id = c.id);
```

(!) Bucketing:

Map joins can take advantage of bucketed tables, since a mapper working on a bucket of the left table needs to load only the corresponding buckets of the right table to perform the join. The syntax for the join is the same as for the in-memory case shown earlier; however, you also need to enable the optimization with the following:

```
SET hive.optimize.bucketmapjoin=true;
```

HiveQL: Queries - Subqueries

#Hive has limited support for subqueries, permitting a subquery in the **FROM** clause, in a **WHERE** clause or inside an **IN** or **EXISTS** statement.

```
SELECT t3.col
FROM (
  SELECT a+b AS col
  FROM t1
  UNION ALL
  SELECT c+d AS col
  FROM t2
) t3
```

```
SELECT A
FROM T1
WHERE EXISTS (
SELECT B FROM T2
WHERE T1.X = T2.Y)
```

```
SELECT *
FROM A
WHERE A.a IN
(SELECT foo FROM B);
```

HiveQL: ORDER BY

```
# ORDER BY: it performs a total Ordering of the query result set. This means that all the data
is passed through a single reducer. May take long time to execute.
> SELECT s.year, s.id, s.price FROM stock s ORDER BY s.year ASC, s.price DESC

# SORT BY: it performs a local Ordering of the query result set. This means that all the data
is passed through a single reducer. May take long time to execute.
> SELECT s.year, s.id, s.price FROM stock s SORT BY s.year ASC, s.price DESC

# DISTRIBUTED BY WITH SORT BY: controls how map output is divided among reducers.
# We can use DISTRIBUTE BY to ensure that the records for each stock symbol go to the same
reducer, then use SORT BY to order the data the way we want.
> SELECT s.ymd, s.symbol, s.price_close
> FROM stocks s
> DISTRIBUTE BY s.symbol
> SORT BY s.symbol ASC, s.ymd ASC;
```

Hive: INDEXES

```
CREATE INDEX index_name
ON TABLE base_table_name (col_name, ...)
AS 'index.handler.class.name'
[WITH DEFERRED REBUILD]
[IDXPROPERTIES (property_name=property_value, ...)]
[IN TABLE index_table_name]
[PARTITIONED BY (col_name, ...)]
[
   [ ROW FORMAT ...] STORED AS ...
   | STORED BY ...
]
[LOCATION hdfs_path]
[TBLPROPERTIES (...)]
[COMMENT "index comment"]
```

- Improve the speed of query lookup on certain columns of a table
- Additional processing to create the index and disk space to store the index
- Only single-table indexes are supported.
- By default, index partitioning matches the partitioning of the base table.For example, a table may be partitioned by date+region even though the index is partitioned by date alone

Hive: Compression & FileFormat

Compression

↑Save disk storage. Reduce traffic in I/O.
↓CPU Overhead.

Trade-Off between Compression Codes:
-> Speed to de/compress -> Space on Disk
(!) Important: Splittable

Compression Codecs

- LZO
- Snappy
- Bzip2 (!)
- LZO(!)

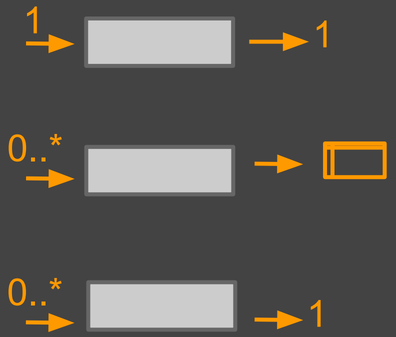
File Formats:

❑ Default: Text format
❑ Sequence Files
❑ RCFile

CONVERT FROM ONE FORMAT TO ANOTHER:
When a query SELECTs from one table and INSERTs into another, Hive uses the metadata about the tables and handles the conversion automatically.

Hive - Functions

- Standard Functions (UDF, or just DF)
- Table Generating Function (UDTF)
- Aggregation Function (UDAF)



- Macros

```
hive> CREATE TEMPORARY MACRO SIGMOID (x DOUBLE) 1.0 / (1.0 + EXP(-x));
hive> SELECT SIGMOID(2) FROM src LIMIT 1;
```

Hive - Meta comandos

```
> show functions;
```

List functions and available operations

```
> describe function [function name]
```

Shows a description of the function

```
> describe function extended [function name
```

Show a detailed description of the function

PrestoDB

Presto is a distributed SQL query engine optimized for ad-hoc analysis at interactive speed. It supports standard ANSI SQL, including complex queries, aggregations, joins, and window functions.

The execution model of Presto is fundamentally different from Hive/MapReduce. Hive translates queries into multiple stages of MapReduce tasks that execute one after another. Each task reads inputs from disk and writes intermediate output back to disk. In contrast, the Presto engine does not use MapReduce. It employs a custom query and execution engine with operators designed to support SQL semantics. In addition to improved scheduling, all processing is in memory and pipelined across the network between stages. This avoids unnecessary I/O and associated latency overhead. The pipelined execution model runs multiple stages at once, and streams data from one stage to the next as it becomes available. This significantly reduces end-to-end latency for many types of queries.

Hive - Good Practices

Partitions	Large files rather than smaller ones
De-normalization	Strict Mode
Compression	Use indexes
Parallelization	Reuse JVM
Optimize Joins	Be carefull with the sintax in your queries.
Tune number of reducers	

Source- Hive Programming chapter 10

Hive - Exercise

Column-wise Variance (s^2) of a matrix

Given a csv file without headers, calculate the sample variance (s^2) of each column.
(http://en.wikipedia.org/wiki/Variance)

$$s^2 = \frac{1}{(N-1)} \sum_{i=1}^{N} (x_i - \bar{x})^2$$

HDFS DataSet path: `/user/hadoop/mapreduce/data/matrix`

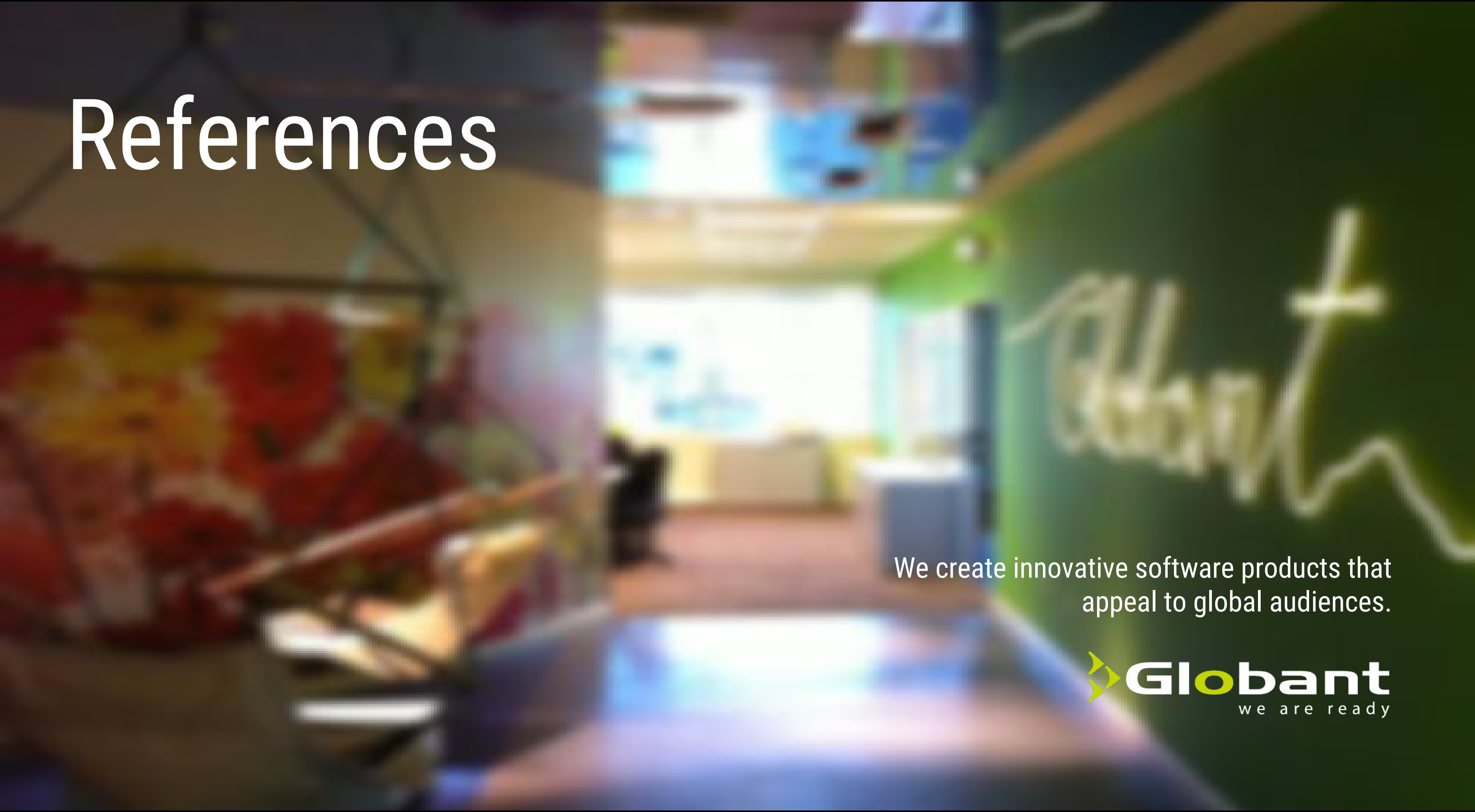
¡¡¡DON'T DOWNLOAD IT FROM THE CLUSTER $$$$!!!

hint 1: Suggested output: sampleVariance1<tab>sampleVariance2<tab>sampleVariance3...
Example:

```
135.6  2.2  536.9  …
```

hint 2: Assume the file has only numeric values and no entries are missing (no NULLs or empty).

References



We create innovative software products that appeal to global audiences.

Globant
we are ready

References

- **Apache Hive Documentation.**- https://cwiki.apache.org/confluence/display/Hive/Home
- **Hadoop: The Definitive Guide, 2nd Edition** (Chapter 11). O'Reilly Media / Yahoo Press - Online @ Globant’s Big Data Training Site
- **Programming Hive**. O'Reilly Media - Online @ Globant’s Big Data Training Site

Thanks