数据分析与挖掘

Hive数据类型系统

- 原子类型
  - 数值，时间日期，字符
  - 布尔
  - 二进制
- 复杂类型
  - Array
  - Map
  - Struct
  - Union

Hive数据类型系统---原子类型

类型	大小	举例
TINYINT	**1 byte 有符号整型**	**20**
SMALLINT	**2 byte 有符号整型**	**20**
INT	**4 byte 有符号整型**	**20**
BIGINT	**8 byte 有符号整型**	**20**
BOOLEAN	布尔类型	**true**
FLOAT	单精度浮点型	**3.14159**
DOUBLE	双度浮点型	**3.14159**
STRING(CHAR, VARCHAR)	字符串	**‘Now is the time’，“for all”**
TIMESTAMP（Date）	整型、浮点型或字符串	**1327882394 (Unixepochseconds), 1327882394.123456789 (Unixepoch seconds plus nanoseconds),**
BINARY	字节数组	
DECIMAL		**从Hive0.11.0开始支持**
CHAR		**从Hive0.13.0开始支持**
VARCHAR		**从Hive0.12.0开始支持**
DATE	**通常用String代替DATE**	**从Hive0.12.0开始支持**

Hive数据类型转换

- Hive开发常用数据类型 String、int、 DECIMAL(整浮点型)（一般设置为decimal（20,3））
- Hive用CAST来显式的将一个类型的数据转换成另一个数据类型。语法cast(value AS TYPE)。如员工表employees，有name、salary等字段；salary是字符串类型的。有如下的查询：

```
SELECT name, salary FROM employees
WHERE cast(salary AS FLOAT) < 100000.0;
```

- salary将会显示的转换成float。如果salary是不能转换成float，这时候cast将会返回NULL。

Hive数据类型转换

- 将浮点型的数据转换成int类型的，内部操作是通过round()或者floor()函数来实现的，而不是通过cast实现
- 只能将BINARY类型的数据转换成STRING类型。如果你确信BINARY类型数据是一个数字类型，可以利用嵌套的cast操作，比如a是一个BINARY，且它是一个数字类型，可以用下面的查询：

SELECT (cast(cast(a as string) as double)) from src;

可将一个String类型的数据转换成BINARY类型。

Hive数据类型系统---原子类型

对于Date类型的数据，只能在Date、Timestamp以及String之间进行转换。

有效的转换	结果
cast(date as date)	返回date类型
cast(timestamp as date)	timestamp中的年/月/日的值是依赖与当地的时区，结果返回date类型
cast(string as date)	如果string是YYYY-MM-DD格式的，则相应的年/月/日的date类型的数据将会返回；但如果string不是YYYY-MM-DD格式的，结果则会返回NULL。
cast(date as timestamp)	基于当地的时区，生成一个对应date的年/月/日的时间戳值
cast(date as string)	date所代表的年/月/日时间将会转换成YYYY-MM-DD的字符串。

Hive数据类型系统---复杂类型

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

类型	解释
ARRAY	ARRAY类型是由一系列相同数据类型的元素组成，这些元素可以通过下标访问，如有一个ARRAY类型的变量fruits，由[‘apple’，‘orange’，‘mango’] 组成，通过fruits[1]来访问元素orange，因为ARRAY类型的下标是从0开始的 **array('John', 'Doe')**
MAP	存储**key/value**对，可通过**[‘key’]**获取每个**key**的值，比如 **‘first’ → ‘John’ and ‘last’ → ‘Doe’ ,**可通过**name[‘last’]**获取**last name。** **map('first', 'John','last', 'Doe')**
STRUCT	与**C/C++**中的结构体类似，可通过 “**.**” 访问每个域的值，比如**STRUCT {first STRING; lastSTRING}**，可通过**name.first**访问第一个 STRUCT可以包含不同数据类型的元素。这些元素可以通过” 点语法” 的方式 来得到所需要的元素比如user 是一个STRUCT类型，那么可以通过user.address得到这个 用户的地址成员。 **struct('John', 'Doe')**

Hive数据类型转换

	boolean	tinyint	smallint	int	bigint	float	double	decimal	string	varchar	timestamp	date	binary
boolean	true	false	false	false	false	false	false	false	false	false	false	false	false
tinyint	false	true	true	true	true	true	true	true	true	true	false	false	false
smallint	false	false	true	true	true	true	true	true	true	true	false	false	false
int	false	false	false	true	true	true	true	true	true	true	false	false	false
bigint	false	false	false	false	true	true	true	true	true	true	false	false	false
float	false	false	false	false	false	true	true	true	true	true	false	false	false
double	false	false	false	false	false	false	true	true	true	true	false	false	false
decimal	false	false	false	false	false	false	false	true	true	true	false	false	false
string	false	false	false	false	false	false	true	true	true	true	false	false	false
varchar	false	false	false	false	false	false	true	true	true	true	false	false	false
timestamp	false	false	false	false	false	false	false	false	true	true	true	false	false
date	false	false	false	false	false	false	false	false	true	true	false	true	false
binary	false	false	false	false	false	false	false	false	false	false	false	false	true

Hive文件编码

Hive中默认的记录和字段分隔符

分隔符	解释
\n	记录间的分割符，默认一行一条记录
^A（“control” A)	列分隔符，通常写成“\001”
^B	ARRAY或STRUCT中元素分隔符，或MAP中key与value分隔符，通常写成“\002”
^C	MAP中key/value对间的分隔符，通常写成“\003”

John Doe^A100000.0^AMary Smith^BTodd Jones^AFederal Taxes^C.2^BState Taxes^C.05^BInsurance^C.1^A1 Michigan Ave.^BChicago^BIL^B60600

Mary Smith^A80000.0^ABill King^AFederal Taxes^C.2^BState Taxes^C.05^BInsurance^C.1^A100 Ontario St.^BChicago^BIL^B60601

Todd Jones^A70000.0^A^AFederal Taxes^C.15^BState Taxes^C.03^BInsurance^C.1^A200 Chicago Ave.^BOak Park^BIL^B60700

Bill King^A60000.0^A^AFederal Taxes^C.15^BState Taxes^C.03^BInsurance^C.1^A300 Obscure Dr.^BObscuria^BIL^B60100

这个例子数据之间的可读性并不是很好，可以使用Hive来读取这些数据。

Hive文件编码

```
{
  "name": "John Doe",
  "salary": 100000.0,
  "subordinates": ["Mary Smith", "Todd Jones"],
  "deductions": {
    "Federal Taxes": .2,
    "State Taxes":    .05,
    "Insurance":      .1
  },
  "address": {
    "street": "1 Michigan Ave.",
    "city": "Chicago",
    "state": "IL",
    "zip": 60600
  }
}
```

这是用户使用默认的分割符后得到的。但是用户可以不用默认的分隔符，而指定使用其他分隔符。当有其他应用程序使用不同规则写数据的时候，这是很必要的。

Hive文件编码

```
CREATE TABLE employees (

  name         STRING,
  salary       FLOAT,
  subordinates ARRAY<STRING>,
  deductions   MAP<STRING, FLOAT>,
  address      STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
COLLECTION ITEMS TERMINATED BY '\002'
MAP KEYS TERMINATED BY '\003'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

明确指定分隔符。

Hive读时模式

对于Hive要查询的数据，有很多种方式对其进行创建，修改，甚至破坏。因此，Hive不会在数据加载时进行验证，而是在查询时进行，这就是读时模式。

传统数据库是写时模式，即数据在写入数据库时对模式进行检查。

数据模型

1 表
2 外部表
3 分区
4 桶

- 分区和分桶可以单独用于表
- 分区可以是多级的
- 分区和分桶可以嵌套使用，但分区必须在分桶前
- 只有桶对应的是文件

Hive数据模型

- Databases：和关系型数据库中的数据库一样
- Tables：和关系型数据库中的表一样
- Partitions(可选) ：一些特殊的列，优化数据的存储和查询
- Buckets (可选)：一种特殊的分区数据组织方式
- Files：实际数据的物理存储单元

Hive数据库

- 创建数据库

hive> create database beihang

- 删除数据库

hive> drop database beihang

当数据中存在表时，需要加CASCADE才能删除。一旦删除成功，数据库在HDFS中的文件夹也被删除

- 查看当前存在的所有数据库

Show databases;

- 查看指定数据库结构

hive> describe database beihang [desc]

- 使用（进入）指定数据库

hive> use beihang

- 查看当前数据库下的所有表

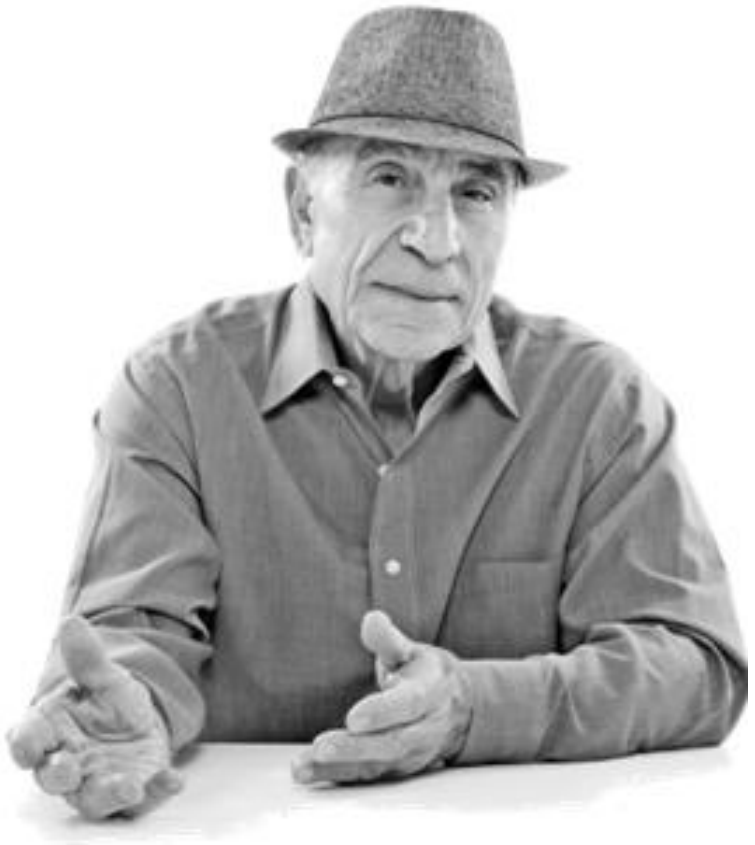
hive> show tables;

Hive表

类似关系型数据库表，每个表在HDFS中都有相应目录来存储表的数据，目录通过${HIVE_HOME}/conf/hive-site.xml配置文件hive.metastore.warehouse.dir属性来配置，默认值/opt/bigdata/hive/warehouse。

如表employees，在HDFS中会创建/opt/bigdata/hive/warehouse/employees目录；employees表所有的表数据（除了外部表）都存放在这个目录中

Hive表数据由两部分组成：数据文件和元数据

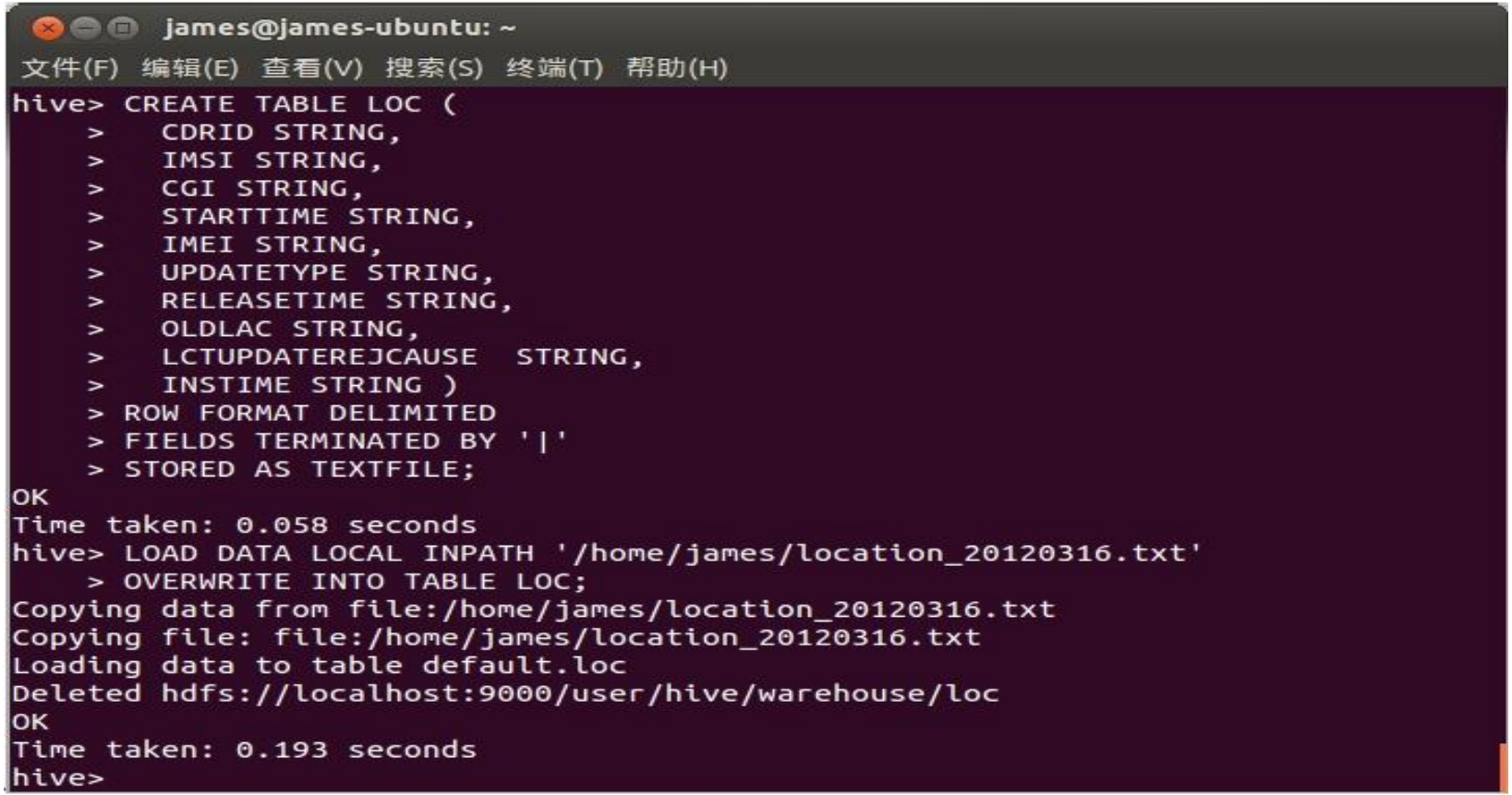


创建表

```
CREATE TABLE IF NOT EXISTS employees (
        name    STRING,
        salary  FLOAT,
        subordinates ARRAY<STRING>,
        decutions       MAP<STRING, FLOAT>,
        address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
COLLECTION ITEMS TERMINATED BY '\002'
MAP KEYS TERMINATED BY '\003'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

- ROW FORMAT DELIMITED：保留关键字
- FIELDS TERMINATED BY：列分隔符
- COLLECTION ITEMS TERMINATED BY ：元素间分隔符
- MAP KEYS TERMINATED BY：key/value对间的分隔符
- LINES TERMINATED BY：行分隔符

创建表


james@james-ubuntu: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
hive> CREATE TABLE LOC (
> CDRID STRING,
> IMSI STRING,
> CGI STRING,
> STARTTIME STRING,
> IMEI STRING,
> UPDATETYPE STRING,
> RELEASETIME STRING,
> OLDLAC STRING,
> LCTUPDATEREJCAUSE STRING,
> INSTIME STRING)
> ROW FORMAT DELIMITED
> FIELDS TERMINATED BY '|'
> STORED AS TEXTFILE;
OK
Time taken: 0.058 seconds
hive> LOAD DATA LOCAL INPATH '/home/james/location_20120316.txt'
> OVERWRITE INTO TABLE LOC;
Copying data from file:/home/james/location_20120316.txt
Copying file: file:/home/james/location_20120316.txt
Loading data to table default.loc
Deleted hdfs://localhost:9000/user/hive/warehouse/loc
OK
Time taken: 0.193 seconds
hive>

创建表

```
CREATE TABLE `employees`(
  `name` string,
  `salary` float,
  `subordinates` array<string>,
  `decutions` map<string,float>,
  `address` struct<street:string,city:string,state:string,zip:int>)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES (
  'colelction.delim'='\u0002',
  'field.delim'='\u0001',
  'line.delim'='\n',
  'mapkey.delim'='\u0003',
  'serialization.format'='\u0001')
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  'hdfs://localhost:9000/user/hive/warehouse/employees'
TBLPROPERTIES (
  'numFiles'='1',
  'numRows'='0',
  'rawDataSize'='0',
  'totalSize'='429',
  'transient_lastDdlTime'='1482424312')
Time taken: 1.589 seconds, Fetched: 26 row(s)
```

查看表

- 查看当前数据库所有表

```
hive> show tables;
```

- 模糊查询查看当前数据库所有匹配的表名

```
hive> show tables '.*s';
```

- 查看建表语句tables

```
hive> show create table wangbh;
```

- 查看表结构

```
hive> desc wangbh;
```

- 删除指定表

```
hive> drop table wangbh;
```

- 创建一个结构与employees一样的新表

```
hive> create table wangbh like employees;
```

删除数据库/表

- DROP DATABASE beihang
- DROP TABLE wangbh
  - 元数据被删除
  - 内部表的数据被删除
  - 外部表的数据不会被删除

修改表

- 将student表的s_sex列名改为s_xingbie
- 将student表的s_sex列名改为s_score，类型改为int，并将它放置在列s_name之后；
- 将student表的s_sex列名改为s_xingbie，并把它放第一列；

```
hive> create table student (s_no int, s_name string, s_sex string);
alter table student change s_sex s_xingbie string;
alter table student change s_sex s_score int after s_name;
alter table student change s_sex s_xingbie string first;
```

插入数据到表

- 加载数据 LOAD DATA---加载数据文件到hive表中

LADO DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE <table_name>
[PARTITION (partcol1=val1, partcol2=val2 ...)]

- INSERT---从别的表中查询出相应的数据并导入到Hive表中

INSERT [OVERWRITE] | [INTO] TABLE <table_name> [PARTITION (partcol1=val1,
partcol2=val2 ...)] select_statement FROM from_statement

- 创建表时加载数据文件

[LOCATION hdfs_path]

- 创建表时查询加载数据 Create Table As Select

LOAD DATA LOCAL INPATH '/hadoop/files/employees.txt' OVERWRITE INTO TABLE employees;

插入数据到表

添加新一列

```
hive> ALTER TABLE invites ADD COLUMNS (new_col2 INT
COMMENT 'a comment');
```

从指定表中选取数据插入到其他表中

```
FROM src
    INSERT OVERWRITE TABLE dest1 SELECT src.* WHERE src.key <
100
    INSERT OVERWRITE TABLE dest2 SELECT src.key, src.value
WHERE src.key >= 100 and src.key < 200
```

删除表

删除表中数据，但要保持表的结构定义

```
hive> dfs -rmr /opt/bigdata/hive/warehouse/records;
```

删除内部表时会删除元数据和表数据文件

删除外部表（external）时只删除元数据

查询表

```
SELECT pv.pageid, u.age FROM page_view p JOIN user u ON
(pv.userid = u.userid) JOIN newuser x on (u.age = x.age);
Select * from uid limit 50;
```

查询表

```
hive> SELECT IMSI, CGI, TIME FROM RESULT;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201206262230_0012, Tracking URL = http://localhost:50030/jobd
etails.jsp?jobid=job_201206262230_0012
Kill Command = /home/james/hadoop/bin/../bin/hadoop job  -Dmapred.job.tracker=lo
calhost:9001 -kill job_201206262230_0012
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2012-06-27 00:46:38,221 Stage-1 map = 0%,  reduce = 0%
2012-06-27 00:46:41,231 Stage-1 map = 100%,  reduce = 0%
2012-06-27 00:46:44,244 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201206262230_0012
MapReduce Jobs Launched:
Job 0: Map: 1   HDFS Read: 538 HDFS Write: 328 SUCESS
Total MapReduce CPU Time Spent: 0 msec
OK
460000722940589 10193-7513      2012-03-15 23:59:35
460023173370082 10188-9376      2012-03-15 23:59:35
460029146542227 10188-9112      2012-03-15 23:59:34
460000940196027 10193-9197      2012-03-15 23:59:35
460020202346902 9417-8743       2012-03-15 23:59:34
460022676514472 10188-9196      2012-03-15 23:59:35
460027157683337 10188-8165      2012-03-15 23:59:35
Time taken: 12.705 seconds
hive>
```

Hive外部表

如果所有处理都需要由Hive完成，那么应该创建表，否则使用外部表

Hive中外部表和表在元数据的组织上相同。

1、实际数据存储不同，外部表数据不是放在自己表所属的目录中，而是存放到别处，这样的好处是如果要删除外部表，该外部表所指向的数据不会被删除的，只会删除外部表对应的元数据；
如果要删除表，该表对应的所有数据包括元数据都会被删除。

2、外部表的创建只有一个步骤，加载数据和创建表同时完成，实际数据存储在创建语句LOCATION制定的HDFS路径中，并不会移动到数据仓库目录中。创建表的操作包含两个步骤：表创建过程和数据加载步骤（这两个过程可以在同一语句中完成）。在数据加载过程中，实际数据会移动到数据仓库目录中。之后的数据访问将会直接在数据仓库目录中完成。

```
hive> truncate table table_name [partition partition_spec];
```

truncate 不能删除外部表！因为外部表里的数据并不是存放在Hive的数据仓库中

创建外部表

某网站日志格式

```
10180,2009-03-01 00:34:44,20090301,1001,12911621
10180,2009-03-01 00:34:46,20090301,0,841842809
10180,2009-03-01 00:34:48,20090301,10180,22395900
10180,2009-03-01 00:34:50,20090301,10180,867110912
```

- 表定义如下：

```
CREATE EXTERNAL TABLE logs(
    domain_id INT ,
    log_time STRING ,
    log_date STRING,
    log_type INT,
    uin BIGINT )
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ',  '
   LOCATION '/opt/bigdata/hivetest/logs';
```

内部表和外部表

表类型	内部表	外部表
语句	create table	create external table
是否移动数据到warehouse	是	否
删除table，是否删除数据	是	否，仅删除元数据
源数据	本地磁盘数据或HDFS数据	已经在HDFS中存在的数据
创建过程和数据加载过程	分别独立完成（也可以在同一个语句中完成，但数据不会移动到数据仓库）	加载数据和创建表同时完成

内部表和外部表

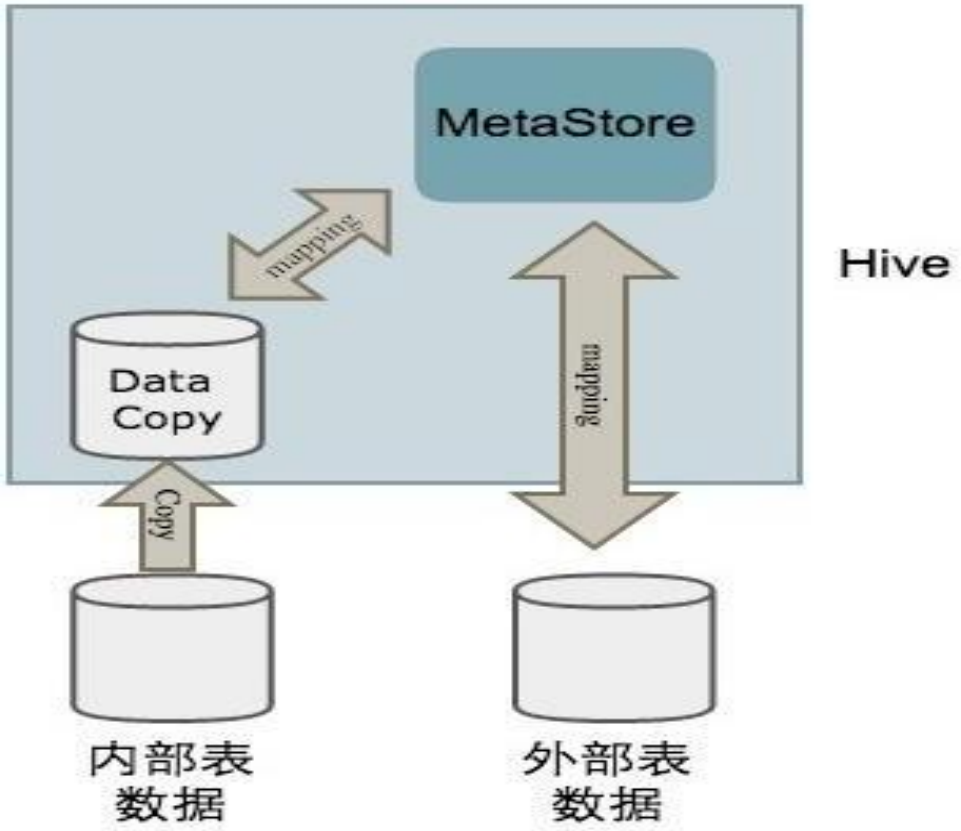
- 内部表对数据拥有所有权，将内部表数据保存在

hive.metastore.warehouse.dir目录下，删除内部表时，相应数据也会被删除

- Hive 对外部表的数据仅仅拥有使用权
- 外部表只有一个过程，加载数据和创建表同时完成

（CREATE EXTERNAL TABLE...LOCATION），实际数据

是存储在LOCATION后面指定的HDFS 路径中，并不会移动到数据仓库目录中



内部表和外部表

- 外部表使用场景：导入hdfs中的源数据
- 内部表使用场景：存放Hive处理的中间表、结果表

- 每天将日志数据传入HDFS，一天一个目录
- Hive基于流入的数据建立外部表，将每天HDFS上的原始日志映射到外部表的天分区中
- 在外部表基础上做统计分析，使用内部表存储中间表、 结果表，数据通过SELECT+INSERT进入内部表

Hive分区

- 分区对应数据库中的相应分区列的一个索引，分区的组织方式和RDB不同。表中的一个分区对应一个目录，分区的数据都存储在对应的目录中。
- htable表中包含的ds和city两个分区，分别对应两个目录
- 对应于ds=20100301，city=beijing的HDFS子目录为：/datawarehouse/htable/ds=20100301/city=beijing;
- 对应于ds=20100301，city=shanghai的HDFS子目录为/datawarehouse/htable/ds=20100301/city=shanghai

Hive分区

- 为减少不必要的暴力数据扫描，可以对表进行分区
- 为避免产生过多小文件，建议只对离散字段进行分区

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

```
...
.../employees/country=CA/state=AB
.../employees/country=CA/state=BC
...
.../employees/country=US/state=AL
.../employees/country=US/state=AK
...
```

```
SELECT * FROM employees
WHERE country = 'US' AND state = 'IL';
```

创建分区表

创建分区表：

hive> create table logs(ts bigint,line string) partitioned by (dt String,country String);

插入分区表数据：

hive> load data local inpath ‘/home/hadoop/input/hive/partitions/file1’ into table logs partition (dt= ‘2001-01-01’,country= ‘GB’);

导入（本地、hdfs）文件到指定的表分区

Load data inpath ‘/data/test/seq1.txt’ into table uid;

Load data inpath ‘/data/test/2013/seq1.txt’ into table uid PARTITION(‘2013’);

LOAD DATA INPATH '/opt/bigdata/myname/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');

```
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;
```

创建分区表

- partitioned by 指定分区字段
- clustered by 对表和分区对某个列进行分桶操作　 sorted by对某个字段进行排序
- row format 指定数据行中字段间的分隔符和数据行分隔符
- stored as 指定数据文件格式：textfile sequence rcfile inputformat (自定义的inputformat 类)
- location 指定数据文件存放的hdfs目录

创建分区表

- order by会做全局排序
- 用order by 记得加 limit
- 尽可能用sort by，每个reduce上的结果有序（即局部有 序）
- distribute by 与 sort by配合使用可以接近全局有序
- 如按省市、时间排序，可以用distribute by 省市 sort by省市，时间

select XXX,XXX from XXX distribute by 省市 sort by 省市 asc，时间 desc

创建分区表

```
CREATE TABLE IF NOT EXISTS employees_part (
        name    STRING,
        salary  FLOAT,
        subordinates ARRAY<STRING>,
        decutions       MAP<STRING, FLOAT>,
        address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
)
PARTITIONED BY (state STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
COLLECTION ITEMS TERMINATED BY '\002'
MAP KEYS TERMINATED BY '\003'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

```
OK
name                    string
salary                  float
subordinates            array<string>
decutions               map<string,float>
address                 struct<street:string,city:string,state:string,zip:int>
state                   string

# Partition Information
# col_name              data_type               comment

state                   string
Time taken: 0.112 seconds, Fetched: 11 row(s)
```

分区表数据导入

- 直接导入数据
  - LOAD DATA LOCAL INPATH '/hadoop/examples/files/employees.txt' OVERWRITE INTO TABLE employees_part PARTITION(state='IL');
- 通过SQL插入
  - INSERT INTO TABLE employees_part PARTITION(state = 'IL') SELECT * FROM employees where address.state='IL';

分区表数据导入

- FROM employees e
  INSERT OVERWRITE TABLE employees_part PARTITION(state = 'IL')
  SELECT * where e.address.state='IL'
  INSERT OVERWRITE TABLE employees_part PARTITION(state = 'CA')
  SELECT * where e.address.state='CA'
  INSERT OVERWRITE TABLE employees_part PARTITION(state = 'NY')
  SELECT * where e.address.state='NY';

- set hive.exec.dynamic.par44on.mode=nonstrict
- FROM employees e
  INSERT OVERWRITE TABLE employees_part

展示分区表

➢查看分区表的分区结构

hive> show partitions wangbh

hive>show partitions wangbh partition （partition_sec=‘partition_value'）

修改表 删除分区

ALTER TABLE page_view DROP PARTITION (dt='2008-08-08', country='us');

ALTER TABLE c02_clickstat_fatdt1 DROP PARTITION (dt='20101202');

增加分区

```
ALTER TABLE page_view ADD PARTITION (dt='2008-08-08', country='us') location
'/path/to/us/part080808' PARTITION (dt='2008-08-09', country='us') location
'/path/to/us/part080809';
ALTER TABLE c02_clickstat_fatdt1 ADD
PARTITION (dt='20101202') location
'/opt/bigdata/hive/warehouse/c02_clickstat_fatdt1/part20101202'
PARTITION (dt='20101203') location
'/opt/bigdata/hive/warehouse/c02_clickstat_fatdt1/part20101203';
```

增加分区

- 给分区表增加分区

hive> alter table <table_name> add partition(par_src=par_value);

- 删除表分区

hive> alter table <table_name> drop partition(par_src=par_value)；

- 对表某一列进行修改（包括列名，列数据类型，列位置，列注释）

hive> alter table <table_name> change [column] col_old_name col_new_name column_type [comment col_comment] [first|after column_name

Hive分桶

- 有一张日志表需要按照日期和用户id来分区，目的是为了加快查询出谁哪天做了什么
- CREATE TABLE weblog (url STRING, sourceIp STRING) PARTITIONED BY (dt STRING, userId STRING);
- 既想加快查询，又避免出现如此多的小文件，bucket出现了
- CREATE TABLE weblog (url STRING, sourceIp STRING userId STRING) PARTITIONED BY (dt STRING) CLUSTERED BY (userId) INTO 32 BUCKETS;

Hive分桶

目的是为了更快！并行

- 插入数据：

FROM raw_logs INSERT OVERWRITE TABLE weblog PARTITION (dt='2016-11-01') SELECT userId, url, sourceIp WHERE dt='2016-11-01';

- 将 userId列分散至 32 个 bucket，首先对 userId 列的值计算 hash，hash 值为 0 放到part-00000；hash 值为 20 放到part-00020
- 产生的目录结构是

/user/hive/warehouse/weblog/dt=20161101/part-00020

Hive分桶

- 对于值较多的字段，可将其分成若干个Bucket
- 可结合CLUSTERED BY与Bucket使用

```
CREATE TABLE weblog (user_id INT, url STRING, source_ip STRING)
PARTITIONED BY (dt STRING)
CLUSTERED BY (user_id) INTO 96 BUCKETS;


SET hive.enforce.bucketing = true;

FROM raw_logs
INSERT OVERWRITE TABLE weblog
PARTITION (dt='2009-02-25')
SELECT user_id, url, source_ip WHERE dt='2009-02-25';
```

分区和分桶

- Hive通常对数据进行全盘扫描，以满足查询条件
- 分区的优势在于利用维度分割数据，Hive只加载需要数据
- 常用的分区方式：按天、地域
- 分区占用数据表中的一个列名，但并不占用表的实际存储空间
- HDFS不适合存储大量小文件，理想分区方案不应该导致过多的分区文件，可以用分桶来解决过多小文件的问题

HQL

- Hive定义了一个类似SQL的查询语言HQL，能够将用户编写的SQL转化为相应的MapReduce程序。 Hive设计目的是让SQL技能良好，但Java技能较弱的分析师可以查询海量数据。
- HQL不足时，Hive允许使用Map/reduce并行计算模型进行复杂数据分析，也允许熟悉MapReduce开发者的开发自定义的mapper和reducer来处理内建的mapper和reducer无法完成的复杂的分析工作。
- 数据定义语句（DDL）
- 数据操作语句（DML）

HQL

特性	SQL	HQL
更新	UPDATE、INSERT、DELETE	INSERT OVERWRITE TABLE
事务	支持	不支持
索引	支持	不支持（有但不建议使用）
延迟	少于一秒	分钟级
数据类型	基本类型、二进制类型	基本类型、数组、MAP，结构体
函数	数百个内置函数	几十个
多表插入	不支持	支持

HQL数据定义语言DDL

- 常规的DDL
  - Create/Drop/Alter/Use Database
  - Create/Drop/Truncate Table
  - Alter Table/Par44on/Column
  - Create/Drop/Alter View
  - Create/Drop/Alter Index
  - Create/Drop Func4on
  - Show
  - Describe
- Hive独特的DDL
  - Create/Drop Macro
  - Reload Func4on
- Hive不支持的DDL

HQL数据操作语句（DML）

- DML: Load, Insert, Update, Delete
- Select
  - Group By
  - Joins
  - Sort/Distribute/Cluster/Order By
  - Transform and Map---Reduce Scripts
  - Operators and User---Defined Func4ons (UDFs)
  - Union
  - Lateral View
- Explain

HQL DML

- 常规的DML
  - Select
    - Where
    - Group/Order By
    - Joins
    - Union
  - Delete
  - Insert
- Hive不支持的DML
  - Insert into ... values(?,?)
  - 非等值Join
- Hive独特的DML
  - Select
    - Sort/Distribute/Cluster By
    - Transform
    - Lateral View
  - Insert
    - 分区插入/动态分区插入
    - 多表插入

Select

```
[WITH CommonTableExpression (, CommonTableExpression)*]
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
    FROM table_reference
    [WHERE where_condi4on]
    [GROUP BY col_list]
    [ORDER BY col_list]
    [CLUSTER BY col_list
    | [DISTRIBUTE BY col_list] [SORT BY col_list] ]
  [LIMIT number]
```

- with q1 as (select key from src where key = '5') select * from q1;
- with q1 as (select * from src where key= '5') from q1 select *;

Join

- map join/broadcast join
- reduce join/shuffle join/common join
- sort merge bucket join

- left semi join

- left outer join
- right outer join
- full outer join

Join

- Join（仅支持等值连接）
  - INNER JOIN等值连接，只返回两个表中联接字段相等的行
  - LEFT OUTER JOIN左联接，返回包括左表中的所有记录和右表中联接字段相等的记录
  - RIGHT OUTER JOIN右联接，返回包括右表中的所有记录和左表中联接字段相等的记录
  - FULL OUTER JOIN包括两个表的join结果，左边在右边中没找到的结果，右边在左边没找到的结果
  - LEFT SEMI-JOIN是 IN/EXISTS 子查询的一种更高效的实现，只打印出了左边的表中的列，规律是如果主键在右边表中存在，则打印，否则过滤掉
  - Map-side Joins
- 不支持非等值的连接

Join

JOIN类型	优 点	缺 点	适用场景
REDUCE JOIN	可以完成各种JOIN操作	耗时长，占用更多网络资源	任何表大小，没有时间要求
MAP JOIN	可以在map端完成JOIN操作，执行时间短	待连接的两个表必须有一个“小表”,“小表”必须加载内存	事实表和维表
SORT MERGE BUCKET JOIN	转换为小表与小表join，执行 时间短，可以做全连接，几乎不受内存限制	表必须分桶，而且桶内数据有序	两个大表，提前分桶排序

两种分布式Join算法

- Map-side Join（Broadcast join）
  - Join操作在map task中完成，因此无需启动reduce task；
  - 适合一个大表，一个小表的连接操作
  - 思想：小表复制到各个节点上，并加载到内存中；大表分片，与小表完成连接操作
- Reduce-side Join（shuffle join）
  - Join操作在reduce task中完成；
  - 适合两个大表连接操作
  - 思想：map端按照连接字段进行hash，reduce 端完成连接操作

两种分布式Join算法

- Map-side Join（Broadcast join）

```
SELECT /*+ MAPJOIN(b) */ a.key, a.value
FROM a join b on a.key = b.key
```

```
SELECT a.* FROM a JOIN b ON (a.id =b.id AND a.department = b.department)
SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key2)
SELECT a.val, b.val FROM a LEFT OUTER JOIN b ON (a.key=b.key) WHERE a.ds='2009-07-07'AND b.ds='2009-07-07'
```

两种分布式Join算法

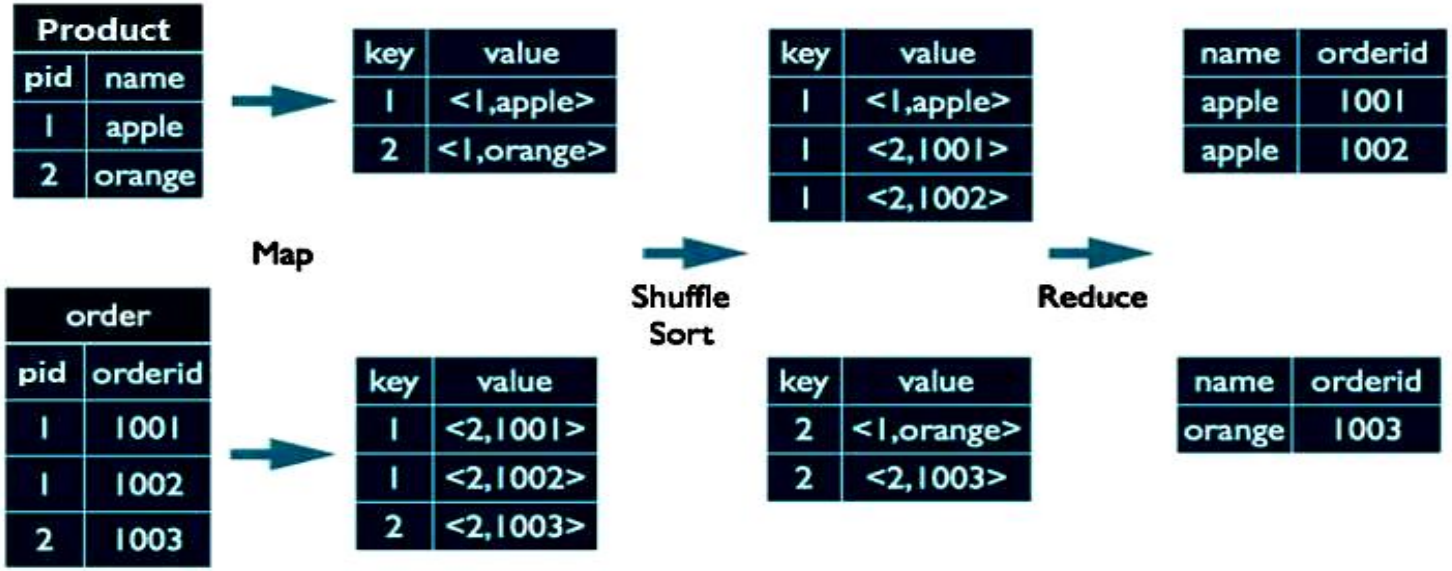
- LEFT SEMI JOIN（左半连接）

```
SELECT a.key, a.value
FROM a
WHERE a.key in
 (SELECT b.key
  FROM B);
```

```
SELECT a.key, a.val
FROM a LEFT SEMI JOIN b on (a.key = b.key)
```

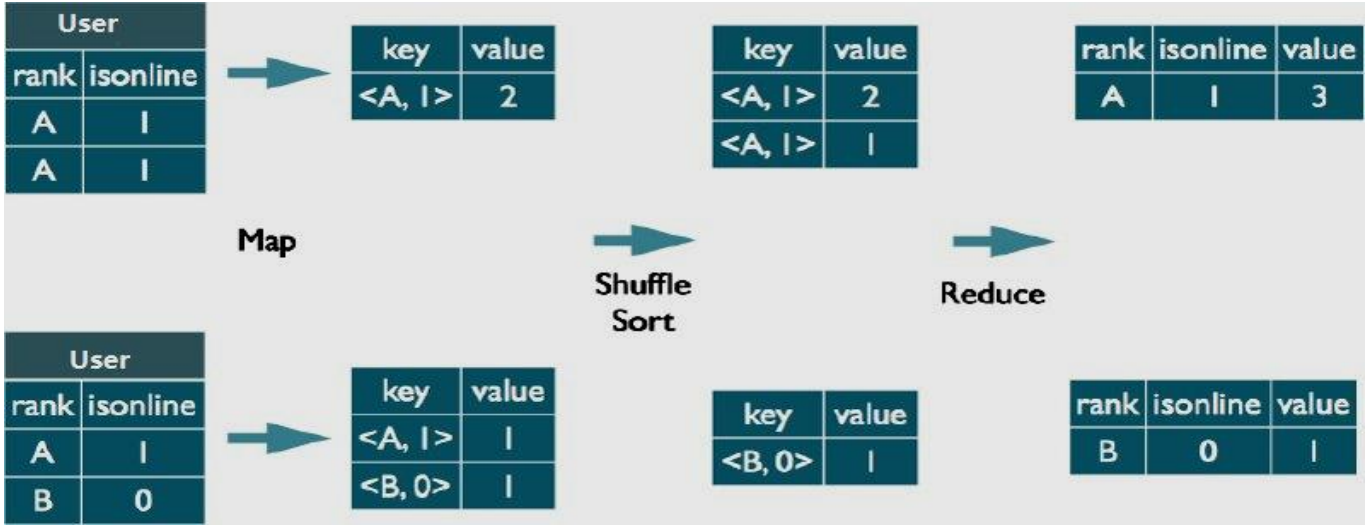
Join

- select product.name, order.orderid from order join product on order.pid = product.pid;



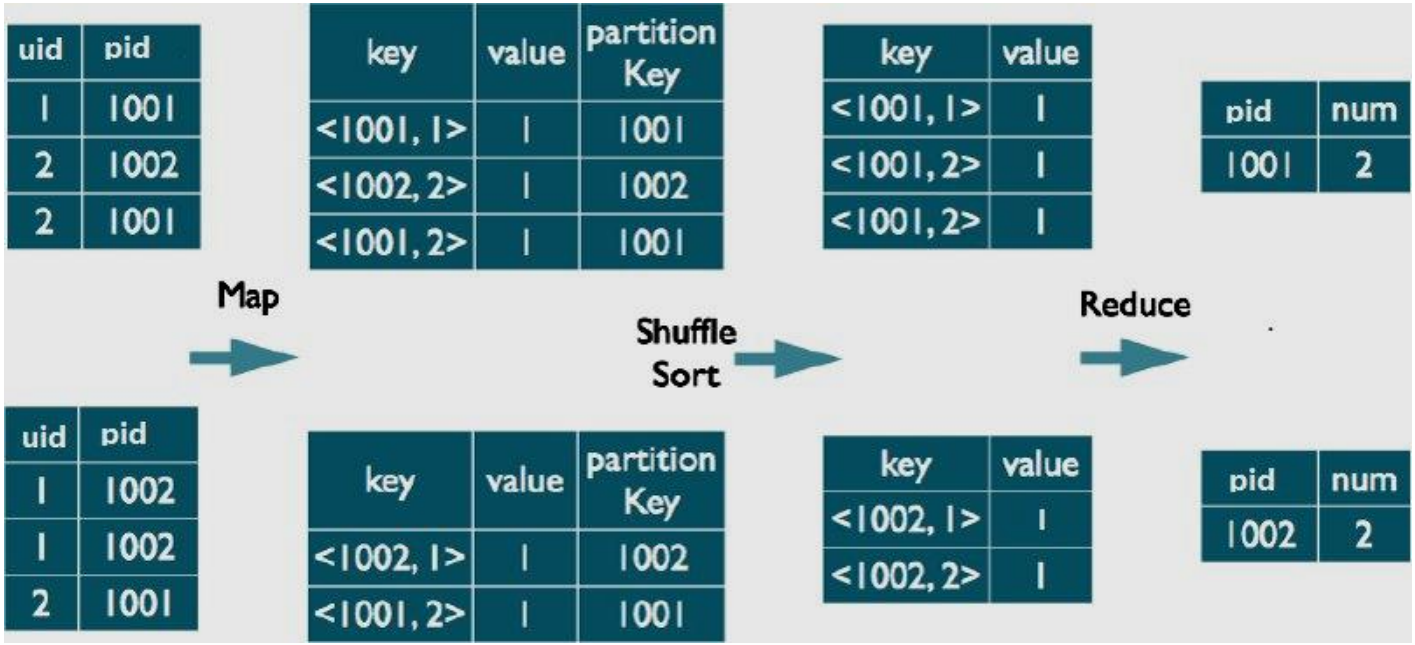
Group By

- SELECT col1 FROM src GROUP By col2
- SELECT col1 FROM t1 GROUP BY col1 HAVING SUM(col2) > 10
- select rank, isonline, count(*) from user group by rank, isonline;



Distinct

- select pid, count(distinct uid) num from order group by pid;



Order By与Sort By

- Order by
  - 启动一个reduce task
  - 数据全局有序
  - 速度可能会非常慢
  - Strict模式下，必须与limit连用
- Sort by
  - 可以有多个reduce task
  - 每个Reduce Task内部数据有序，但全局无序
  - 通常与distribute by

UNION

- SELECT * FROM(select_statement UNION ALL select_statement) bonc
- union all时必须在子查询括号外边加别名（ bonc)

Transform语法（Streaming）

- Hive允许使用任意语言编写Mapper和Reducer，并嵌到HQL中使用
- 实现机制类似于Hadoop Streaming，在Java进程中额外启动一个线程运行脚本/二进制程序，并通过标准输入输出进行数据传递
- 使用操作符TRANSFORM

Transform语法（Streaming）

- 解析一定key/value格式的数据，并对其格式化

```
k1=v1,k2=v2
k4=v4,k5=v5,k6=v6
k7=v7,k7=v7,k3=v7
```

```
k1      v1
k2      v2
k4      v4
k5      v5
k6      v6
k7      v7
k7      v7
k3      v7
```

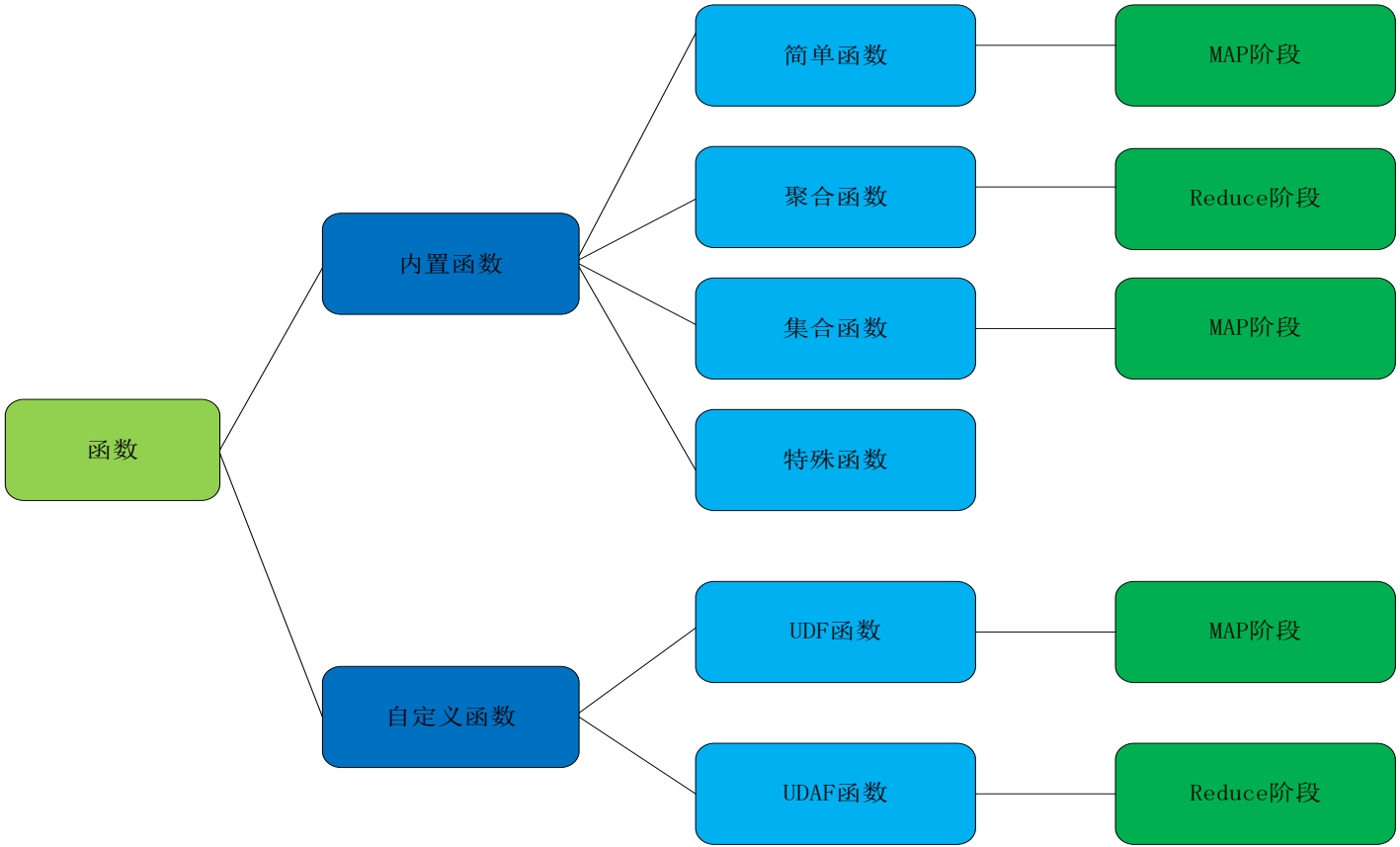
- 编写一下perl脚本:

```
#!/usr/bin/perl
while (<STDIN>) {
    my $line = $_;
    chomp($line);
    my @kvs = split(/,/, $line);
    foreach my $p (@kvs) {
        my @kv = split(/=/, $p);
        print $kv[0] . "\t" . $kv[1] . "\n";
    }
}
```

- 使用该脚本:

```
hive> ADD FILE /home/dongxicheng/split_kv.pl;
hive> SELECT TRANSFORM (line)
> USING 'perl split_kv.pl'
> AS (key, value) FROM kv_data;
```

Hive函数


函数
内置函数
简单函数
聚合函数
集合函数
特殊函数
MAP阶段
Reduce阶段
MAP阶段
自定义函数
UDF函数
UDAF函数
MAP阶段
Reduce阶段

Hive函数

- **简单函数（数据计算粒度-单条记录）**

关系运算、数学运算、逻辑运算、数值计算、类型转换、日期函数、条件函数、字符串函数

- **聚合函数（数据计算粒度-多条记录）**

sum()-求和、count()-求数据量、distinct-求不同的数值、min()-求最小值、max()-求最小值

- **集合函数**

复合类型构建、复杂类型访问、复杂类型长度

- **特殊函数**

窗口函数，分析函数等

Hive函数

- 查看hive支持的函数

hive> show functitions;

- 模糊查看hive支持的函数

hive> show functitions like '*关键字* ‘;

- 查看函数的[扩展]结构[EXTENDED]

hive> desc functitions <function_name> concat;

Hive函数

- 多行转一行

id	name
1	zhangsan
2	lisi
3	wangwu
4	zhaoliu

name	addr
zhangsan	beijing
zhangsan	shanghai
lisi	tianjin
lisi	hefei
wangwu	nanjing
zhaoliu	guangzhou

1	zhangsan	beijing,shanghai
2	lisi	tianjin,hefei
3	wangwu	nanjing
4	zhaoliu	guangzhou

Hive函数

- 一行转多行

1	zhangsan	beijing,shanghai
2	lisi	tianjin,hefei
3	wangwu	nanjing
4	zhaoliu	guangzhou

1	zhangsan	beijing
1	zhangsan	shanghai
2	lisi	tianjin
2	lisi	hefei
3	wangwu	nanjing
4	zhaoliu	guangzhou

用户自定义函数

- UDF：扩展HQL能力的一种方式
- 三种UDF：
  - 普通UDF（1对1）
  - UDAF：用户自定义聚集函数（多对1）
  - UDTF：用户自定义产生表函数（1对多）
  - UDF作用于单个数据行，输出一个数据，如字符处理函数
  - UDAF作用于多个数据行，输出一个数据，如count，sum函数
  - UDTF作用于单个数据行，输出多个数据行支持用户使用java自定义开发UDF函数。
  - Hive streaming：支持用户在hive QL语句中嵌入自定义的streaming处理脚本。

用户自定义函数

- 函数相关操作：
  - SHOW FUNCTIONS;
  - DESCRIBE FUNCTION concat;
  - DESCRIBE FUNCTION EXTENDED concat;
  - SELECT concat(column1,column2) AS x FROM table;

用户自定义函数（UDF）

- 编写复杂的UDF、UDAF和UDTF：
  - 继承类GenericUDF
  - 实现initialize、evaluate等函数
- 相关示例：
  - UDAF：

http://svn.apache.org/repos/asf/hive/branches/branch-0.13/ql/src/java/org/apache/hadoop/hive/ql/udf/generic/GenericUDAFAverage.java

  - UDTF：

http://svn.apache.org/repos/asf/hive/branches/branch-0.13/ql/src/java/org/apache/hadoop/hive/ql/udf/generic/GenericUDTFExplode.java

创建自定义函数

- 自定义一个Java类
- 继承UDF类
- 重写evaluate方法
- 打Jar包
- Hive执行add jar
- Hive执行创建模板函数
- Hql中使用（当前会话）

Lateral View

pageid	add_list
front_page	[1, 2, 3]
contact_page	[3, 4, 5]

```
SELECT pageid, adid
FROM pageAds LATERAL VIEW explode(adid_list) adTable AS adid;
```

pageid(string)	adid(int)
front_page	1
front_page	2
front_page	3
contact_page	3
contact_page	4
contact_page	5

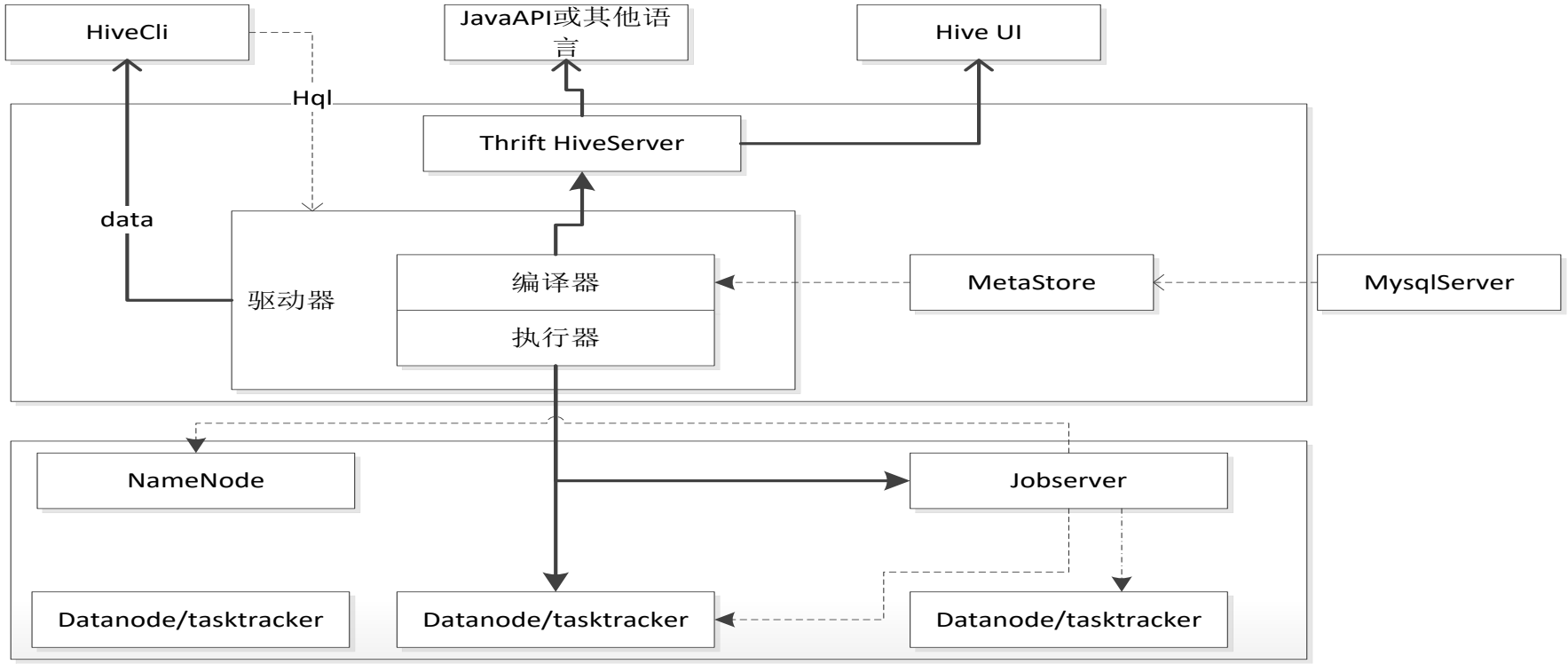
Explain

- 语法：EXPLAIN [EXTENDED] query
- 作用：查看Query的执行计划
- 例子：

```
EXPLAIN FROM src INSERT OVERWRITE TABLE dest_g1
SELECT src.key, sum(substr(src.value,4)) GROUP BY src.key;
```

- 输出：
  - 抽象语法树
  - Stage划分以及依赖
  - Stage的详细执行计划

Hive结构


HiveCli
JavaAPI或其他语言
Hive UI
Hql
Thrift HiveServer
data
驱动器
编译器
执行器
MetaStore
MysqlServer
NameNode
Jobserver
Datanode/tasktracker
Datanode/tasktracker
Datanode/tasktracker

Hive安装

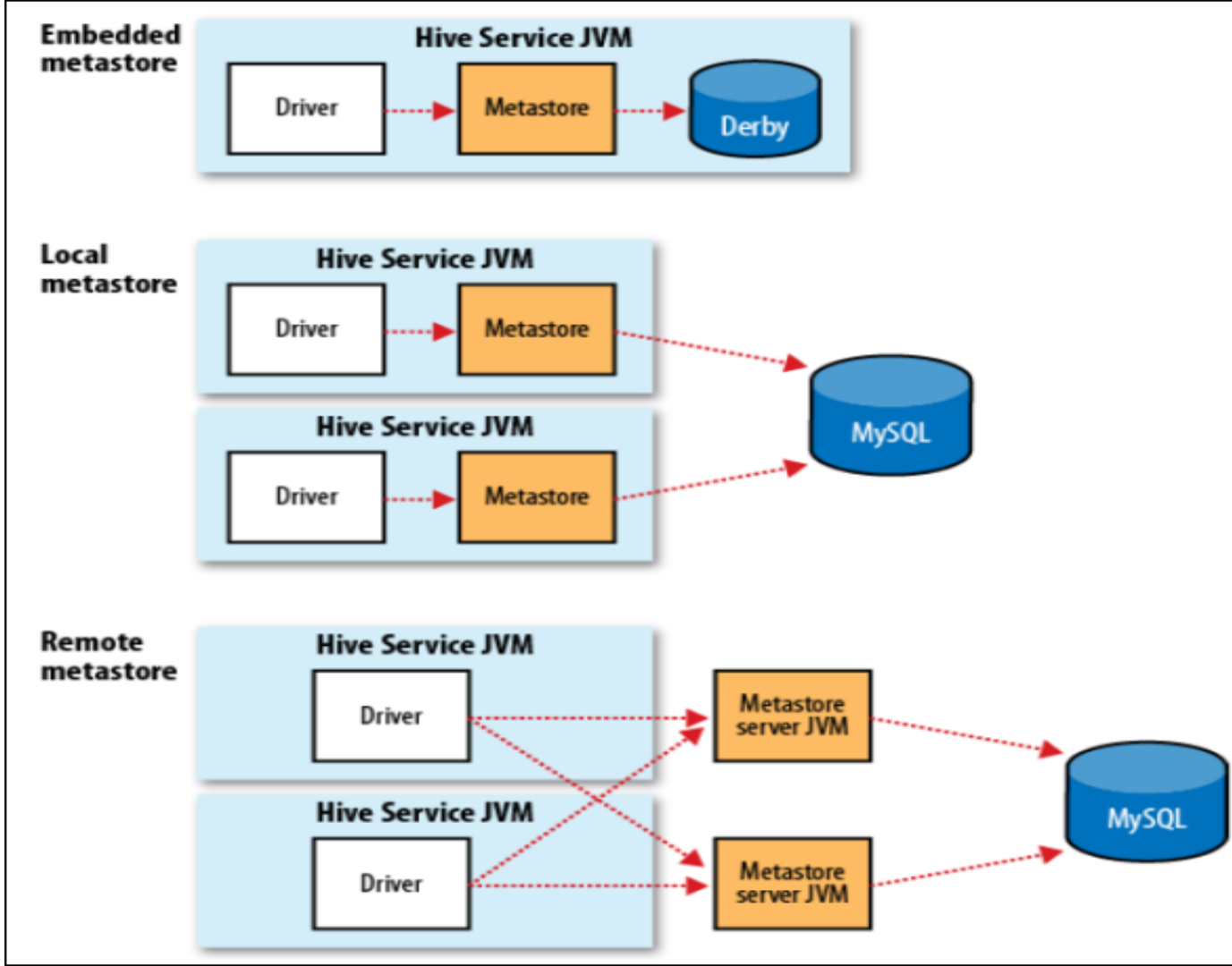
安装好Hadoop集群

hive将元数据存储在RDBMS中。hive安装在namenode上。

内嵌模式：元数据保持在内嵌内存数据库Derby模式，只允许一个会话连接

本地独立模式：在本地安装Mysql，把元数据放到Mysql内

远程模式：元数据放置在远程的Mysql数据库，在本地模式的基础上修改hive-site.xml文件，设置hive.metastore.local为false，指向远程mysql数据库



MySQL安装

- 所有操作以hadoop身份运行，在namenode上先安装mysql。

下载MySQL

rpm –ivh MySQL-server-community-6.0.11-0.rhel5.x86_64.rpm

rpm –ivh MySQL-client-community-6.0.11-0.rhel5.x86_64.rpm

简单安装 yum install mysql-*

启动MySQL

/sbin/service mysql start

登陆mysql

mysql -uroot –p

MySQL安装

在丢失MySQL root密码的时候，可以这样

```
mysqld_safe --skip-grant-tables&
mysql -u root mysql
mysql> UPDATE user SET password=PASSWORD("new password")
WHERE user='root';
mysql> FLUSH PRIVILEGES;
```

MySQL安装

- 添加hive用户名和密码，使用hive口令从任何主机连接到mysql服务器

GRANT ALL PRIVILEGES ON *.* TO 'hive'@'%' IDENTIFIED BY 'hive' WITH GRANT OPTION ;

允许用户hive从ip为192.168.1.%的主机连接到mysql服务器并使用hive作为密码

GRANT ALL PRIVILEGES ON *.* TO 'hive'@'192.168.1.%' IDENTIFIED BY 'hive' WITH GRANT OPTION;

#执行刷新

mysql>FLUSH PRIVILEGES

#使修改生效

[root@master ~]# mysql -uhive -phive -h192.168.1.100

hive基础环境准备

- 下载解压hive安装包，只在namenode上安装。
- 配置安装目录,解压，以hadoop用户身份执行：

```
mkdir -p  /opt/bigdata/hive/
cp hive-0.9.0.tar.gz /opt/bigdata/
cd /opt/bigdata/
tar -xzvf hive-0.9.0.tar.gz
cp * ../hive
```

- 下载hadoop jdbc连接mysql-connector-java-5.1.24.tar.gz
- 解压后cp mysql-connector-java-5.1.20-bin.jar /opt/bigdata/hive/lib/

Hive配置

- 配置环境参数hive-env.sh

cp /opt/bigdata/hive/hive-env.sh.template
/opt/bigdata/hive/hive-env.sh

export HADOOP_HEAPSIZE=256 (默认1024，虚拟机内存比较小一定要改掉)

HADOOP_HOME=/opt/bigdata/ hadoop/

export HIVE_CONF_DIR=/opt/bigdata/ hive/conf

Hive配置

- 配置jdbc连接

vi /opt/bigdata/ hive-1.3.0/hive-site.xml

```
<property>
  <name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:mysql://master1:3306/hive?createDatabaseIfNotExist=true</value>
  <description>JDBC connect string for a JDBC metastore</description>
</property>
```

hive-site.xml,主要配置项：

hive.metastore.warehouse.dir：（HDFS上的）数据目录

hive.exec.scratchdir：（HDFS上的）临时文件目录

hive.metastore.warehouse.dir默认值是/opt/bigdata/hive/warehouse

hive.exec.scratchdir默认值是/tmp/hive-${user.name}

以上是默认值，暂时不改。

创建元数据

hive默认使用内存数据库derby存储元数据，使用时不需要修改任何配置，缺点：hive server重启后所有的元数据都会丢失。

hive还执行mysql、oracle等任何支持JDBC连接方式的数据库来存储元数据，需要修改相应的配置项

注意：要将相应的jdbc的包拷贝到hive的lib目录下

mysql -uhive –phive

#创建Hive metastore

create database hive;

创建元数据

```
<property>
  <name>hive.metastore.local</name>
  <value>false</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:mysql://db1.mydomain.pvt/hive_db?createDatabaseIfNotExist=true</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionDriverName</name>
  <value>com.mysql.jdbc.Driver</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionUserName</name>
  <value>database_user</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionPassword</name>
  <value>database_pass</value>
</property>
```

Hive shell

- 启动hive cli

/opt/bigdata/ hive/bin/hive

/opt/bigdata/ hive/bin/hive --config /opt/bigdata/hive/conf

- 以脚本启动，命令行模式，执行一个sql语句，适合短语句调用

./hive -e "select * from test.test_text limit 30 “

Hive shell

- 读取查询文件查询方式不进入交互模式

#编辑一个HQL 文件 vi test.q

use test;

select * from test_text limit 30;

/opt/bigdata/ hive/bin/hive -f test.q

多语句传递参数 –d，动态的给hive-script.sql脚本传递参数

hive –f /home/hadoop/hive _script.sql -d v_month=201512

select * from tmpdb.test where month_part = '${v_month}'

直接重定向到文本中

hive [-S]-e "select * from <table_name>;">/home/hadoop/test.txt

直接重定向到本地文件系统中，hive的分割符为 '\t'

Hive shell

➢将查询导出为文件

```
set hive.exec.compress.output=false;
insert overwrite [local] directory '指定目录'
row format delimited
fields terminated by '\t'
NULL DEFINED AS ''
SELECT
column1,
column2
FROM TABLE_NAME T
WHERE T.MONTH_PART = '${MONTH_PART}‘ AND T.DAY_PART = '${DAY_PART}'
```

hdfs上hive数据文件一般为压缩文件，数据导出时将 hive.exec.compress.output设置成flase，导出文本格式

OVERWRITE：覆盖指定目录下的文件，不可用into

LOCAL：导出文件到本地文件系统，若无loacl关键字，表示导出文件到Hdfs

Hive客户端

- 客户端
  - Hive CLI
  - Beeline CLI
  - HCatalog CLI
- 命令
  - set <key>=<value>
  - add FILES <filepath>
  - ! <command>
  - .....

变量替换

- Shell中的变量替换
  - tbl=employees
  - hive ---e " describe $tbl"
- Hive中的变量替换
  - hive ------hiveconf tbl=employees ---e 'describe $ {hiveconf:tbl}'
- 关闭变量替换
  - set hive.variable.subs4tute=false;

变量替换

- hiveconf
  - set mapred.reduce.tasks=32
  - ${hiveconf:mapred.reduce.tasks}
- system
  - set system:user.home=/home/hadoop
  - system:user.home
- env
  - set env:JAVA_HOME=/path/to/java_home
  - env:JAVA_HOME

通过JDBC连接Hive

- 启动hive的Thrift Server

hive --service hiveserver

- 新建java项目，然后将hive/lib下的所有jar包和hadoop的核心jar包hadoop-0.20.2-core.jar添加到项目的类路径上。
- 假设hive部署在10.20.151.7机器上conf/hive-default.xml

http://10.20.151.7:9999/hwi/

创建 LZO 压缩表

```
CREATE EXTERNAL TABLE foo (columnA string,
        columnB string )
    PARTITIONED BY (date string)
    ROW FORMAT DELIMITED FIELDS TERMINATED BY "\t"
    STORED AS INPUTFORMAT
    "com.hadoop.mapred.DeprecatedLzoTextInputFormat"
        OUTPUTFORMAT
    "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputF
    ormat"
    LOCATION '/data/dw/uid/';
```

Hive Lzo 文件加载流程

- 压缩lzop uid.txt
- 上传hadoop fs -put uid.txt.lzo /data/dw/uid/
- 创建索引hadoop jar /path/to/your/hadoop-lzo.jar com.hadoop.compression.lzo.DistributedLzoIndexer /data/dw/uid/uid.txt.lzo 1.seq 10 > text

1. hadoop fs -put test /data/text
2. hadoop jar contrib/streaming/hadoop-streaming-1.0.3.jar -mapper 'cat' -reducer 'seq 10000' -numReduceTasks 10 -input /data/text -output /data/test3/
3. CREATE EXTERNAL TABLE test3 (text string)
   LOCATION '/data/test3/'

Hive端口

组件	节点	默认端口	配置
Hive	Metastore	9083	/etc/default/hive-metastore中export PORT=来更新默认端口
Hive	HiveServer	10000	/etc/hive/conf/hive-env.sh中export HIVE_SERVER2_THRIFT_PORT=来更新默认端口

Hive维护

➢通过命令行查看任务执行

yarn application –list

➢登陆web页面查看

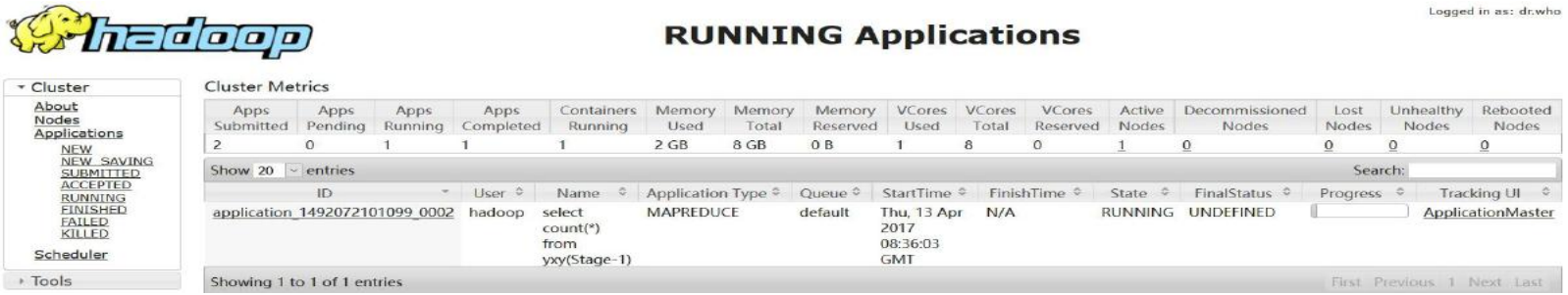
使用主机名： http://BONC:23188/

使用IP地址： http://192.168.1.141:23188/

```
<property>
    <name>yarn.resourcemanager.webapp.address</name>
     <value>BONC:23188</value>
</property>
```

➢KILL指定JOB实例

yarn application -kill application_id



Hive优化

sql语句本身，hive参数，hive底层

- 使用 Partition 减少扫描数量
- 使用Map端Join
- 配置Reduce数量
- xml,json 提取 适用脚本提取,而非使用函数
- 使用 INSERT INTO LOCAL DIRECTORY ‘/home/me/pv_age_sum.dir’ ,而非适用 HiveServer
- 使用 LZO 压缩存储数据
- 适用外部表,而非内部表
- hive.exec.compress.output = false,true
- 适用队列管理任务执行

Hive优化

✱ 使用分区

```
CREATE EXTERNAL TABLE foo (
        columnA string,
        columnB string )
    PARTITIONED BY (site string,day string)
    ROW FORMAT DELIMITED FIELDS TERMINATED BY "\t"
    STORED AS INPUTFORMAT
"com.hadoop.mapred.DeprecatedLzoTextInputFormat"
         OUTPUTFORMAT
"org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat"
    LOCATION '/data/dw/uid/';
```

✱ 合并小文件

hive.merge.mapfiles = true是否和并 Map 输出文件，默认为 True

hive.merge.mapredfiles = false是否合并 Reduce 输出文件，默认为 False

hive.merge.size.per.task = 256*1000*1000合并文件的大小

Hive优化

- Hive是支持索引的，但是很少被使用
- 索引表不会自动rebuild，如果表有数据新增或删除，那么必须手动rebuild索引表数 据
- 索引表本身会非常大
- Hive索引的使用过程比较繁琐

Hive优化

✱ 让hive自行选择左表为较小的表

set hive.auto.convert.join=true;

✱ 同一个sql中不同的job同时运行。(默认false)

set hive.exec.parallel=true;

✱ 执行时设置sql运行时reducer的个数。

相关参数:**mapred.reduce.tasks**

hive.exec.reducers.max

Hive优化

- 排序优化：用sort by ,少用order by
- group by优化

Map端聚合，首先在map端进行初步聚合，最后在reduce端得出最终结果，相关参数：

hive.map.aggr = true是否在 Map 端进行聚合，默认为 True

hive.groupby.mapaggr.checkinterval = 100000在 Map 端进行聚合操作的条目数目

数据倾斜聚合优化，设置参数hive.groupby.skewindata = true，当选项设定为 true，生成的查询计划会有两个 MR Job。第一个 MR Job 中，Map 的输出结果集合会随机分布到 Reduce 中，每个 Reduce 做部分聚合操作，并输出结果，这样处理的结果是相同的 Group By Key 有可能被分发到不同的 Reduce 中，从而达到负载均衡的目的；第二个 MR Job 再根据预处理的数据结果按照 Group By Key 分布到 Reduce 中（这个过程可以保证相同的 Group By Key 被分布到同一个 Reduce 中），最后完成最终的聚合操作。

Hive优化

✱ **join优化：**Join查找操作基本原则：应该将条目少的表/子查询放在 Join 操作符的左边，主要是在 Join 操作的 Reduce 阶段，位于 Join 操作符左边的表的内容会被加载进内存，将条目少的表放在左边，有效减少发生内存溢出错误的几率

Join查找操作中如果存在多个join，且所有参与join的表中其参与join的key相同，则会将所有的join合并到一个mapred程序中。

例：

SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key1）在一个mapre程序中执行join

SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key2) 在两个mapred程序中执行join

Hive优化

Map join的关键在于join操作中的某个表的数据量很小，例：

```
SELECT /*+ MAPJOIN(b) */ a.key, a.value
  FROM a join b on a.key = b.key
```

Mapjoin 的限制是无法执行a FULL/RIGHT OUTER JOIN b

map join相关的hive参数：

hive.join.emit.interval

hive.mapjoin.size.key

hive.mapjoin.cache.numrows

Hive优化

由于join操作是在where操作之前执行，在执行join时，where条件并不能起到减少join数据的作用：

```
SELECT a.val, b.val FROM a LEFT OUTER JOIN b ON (a.key=b.key)
  WHERE a.ds='2009-07-07' AND b.ds='2009-07-07'
```

最好修改为：

```
SELECT a.val, b.val FROM a LEFT OUTER JOIN b
  ON (a.key=b.key AND b.ds='2009-07-07' AND a.ds='2009-07-07')
```

在join操作的每一个mapred程序中，hive都会把出现在join语句中相对靠后的表的数据stream化，相对靠前的变的数据缓存在内存中。当然，也可以手动指定stream化的表：

```
SELECT /*+ STREAMTABLE(a) */ a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key
1) JOIN c ON (c.key = b.key1)
```

Hive优化

✱实现 （not）in
通过left outer join进行查询（假设B表中包含另外的一个字段 key1

select a.key from a left outer join b on a.key=b.key where b.key1 i
s null
通过left semi join 实现 in

SELECT a.key, a.val FROM a LEFT SEMI JOIN b on (a.key = b.key)
Left semi join 的限制：join条件中右边的表只能出现在join条件中

Hive优化

✱Hql使用自定义的mapred脚本

在使用自定义mapred脚本时，关键字MAP REDUCE 是语句SELECT TRANSFORM (...)的语法转换，并不意味着使用MAP关键字时会强制产生一个新的map过程，使用REDUCE关键字时会产生一个red过程。

自定义的mapred脚本可以是hql语句完成更为复杂的功能，但是性能比hql语句差了一些，应该尽量避免使用，如有可能，使用UDTF函数来替换自定义的mapred脚本

Hive锁

➢Hive锁机制配置

```
<property>
<name>hive.support.concurrency</name>
<value>true</value>
<description>Enable Hive's Table Lock Manager Service</description>
</property>
<property>
<name>hive.zookeeper.quorum</name>
<value>hadoop001,hadoop002,hadoop003</value>
<description>Zookeeper quorum used by Hive's Table Lock Manager</description>
</property>
<property>
<name>hive.zookeeper.client.port</name>
<value>2181</value>
</property>
```

Hive锁

LOCKS <TABLE_NAME>;
SHOW LOCKS <TABLE_NAME> EXTENDED;
SHOW LOCKS <TABLE_NAME> PARTITION (<PARTITION_DESC>);
SHOW LOCKS <TABLE_NAME> PARTITION (<PARTITION_DESC>) EXTENDED;
UNLOCK TABLE <TABLE_NAME>;

Hive数据仓库案例

- 有很多web服务器，经常有服务器由于访问量过大而瘫痪，难于分析原因，有什么好的办法吗?

Access_log

1.虚拟主机、2.远程主机、3.空白(E-mail)、4.空白(登录名)、5.请求时间、6.方法+资源+协议、7.状态代码、8.发送字节数

1.远程主机、2.空白(E-mail)、3.空白(登录名)、4.请求时间、5.方法+资源+协议、6.状态代码、7.发送字节数、8.referer 9.user-agent

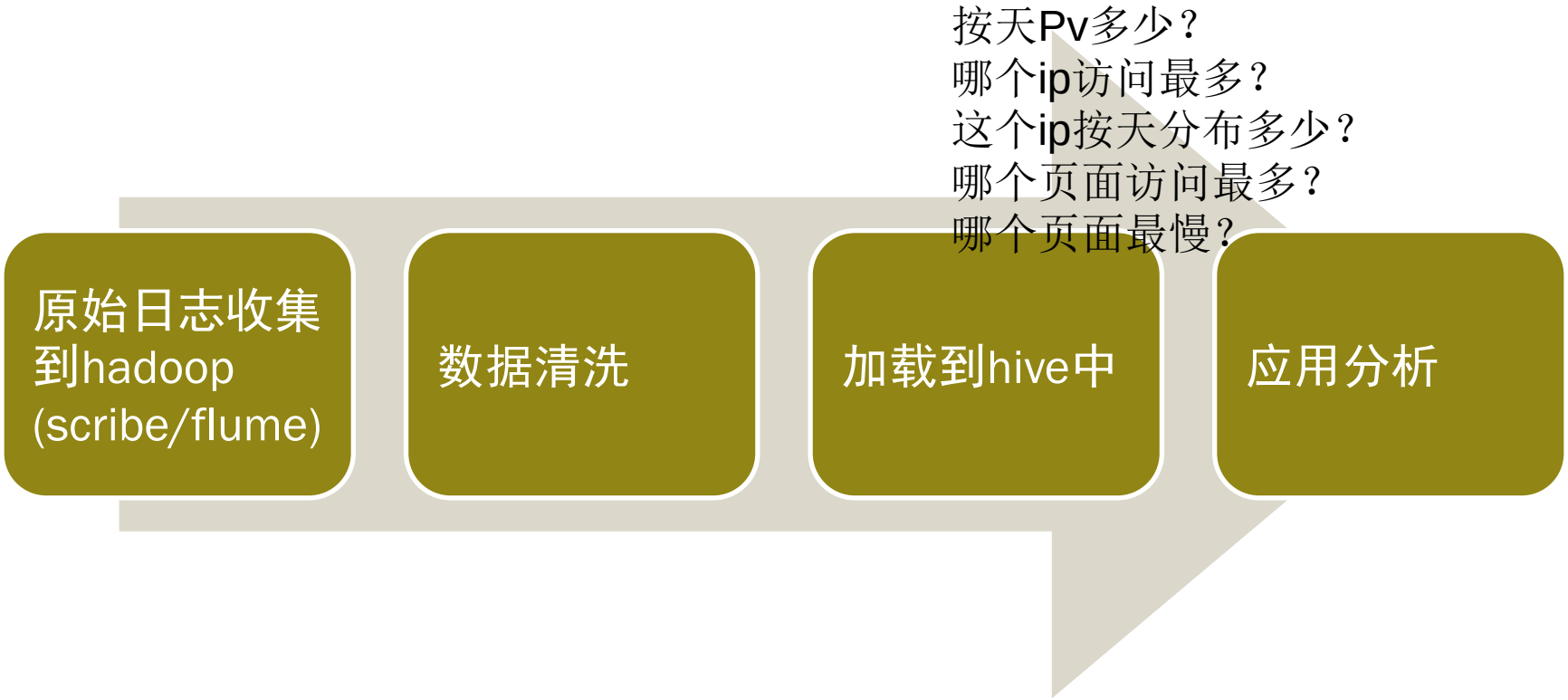
h:host　l:email　u:user登陆名　t:time 访问时间

r:访问的方法，资源，协议

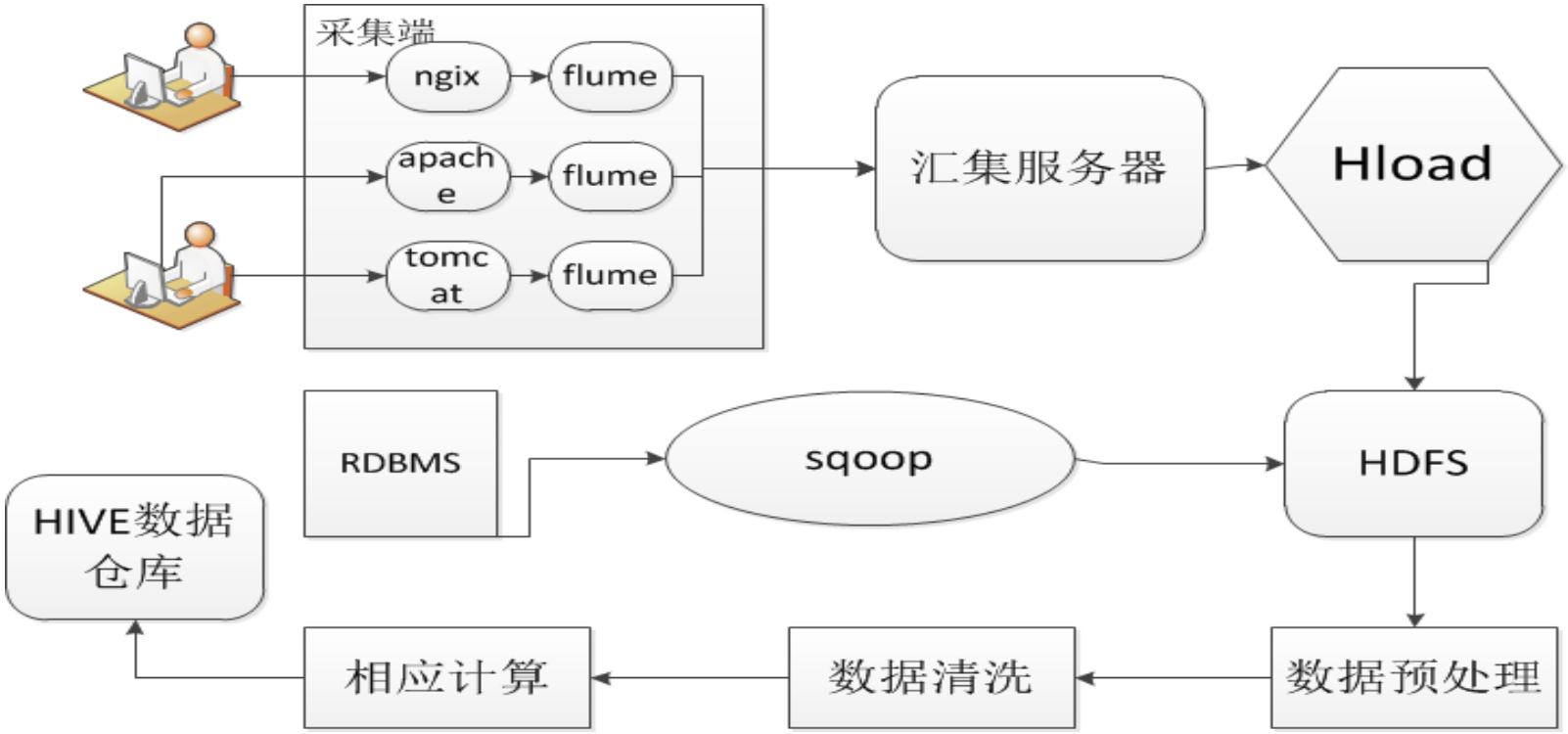
%>s:状态码

%b:字节数

数据处理过程

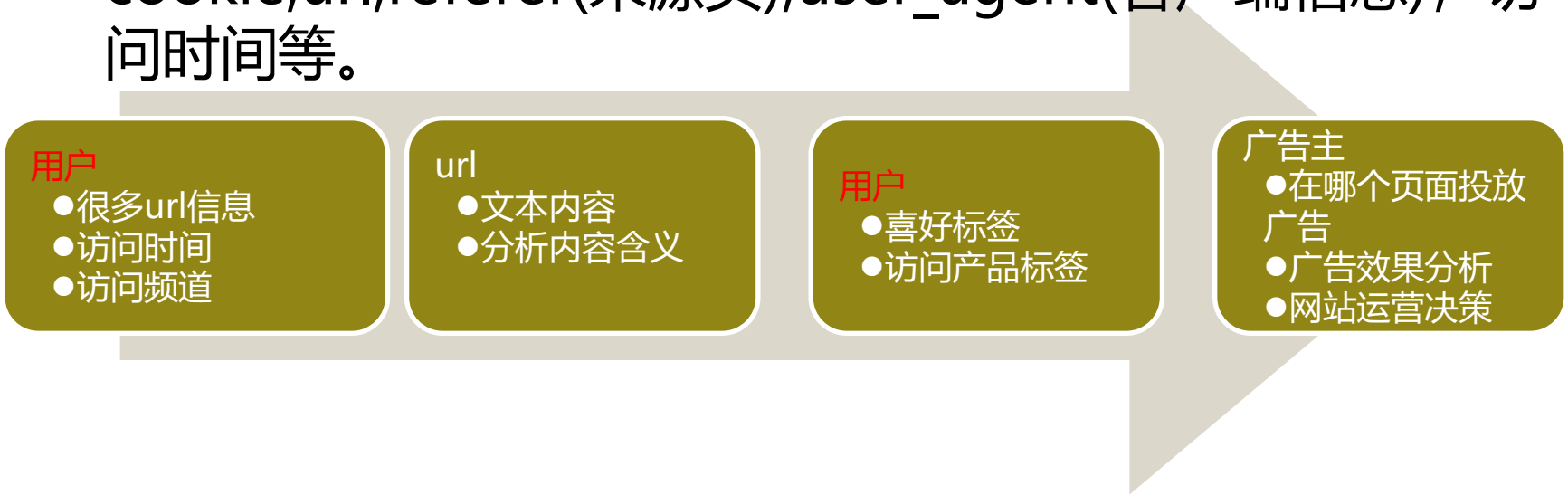

原始日志收集到hadoop (scribe/flume)
数据清洗
加载到hive中
应用分析
按天Pv多少？
哪个ip访问最多？
这个ip按天分布多少？
哪个页面访问最多？
哪个页面最慢？

Hive数据仓库案例


采集端
ngix
flume
apach
e
flume
tomc
at
flume
汇集服务器
Hload
HDFS
sqoop
RDBMS
HIVE数据
仓库
相应计算
数据清洗
数据预处理

Hive精准营销案例

* 我们在访问sina,sohu,ifeng,163等的时候会有大量的访问行为同时也会进入第三方数据服务商或者广告运营平台的手中。收集了大量的这种信息，包含cookie,url,referer(来源页),user_agent(客户端信息)，访问时间等。



Hive精准营销案例

原始数据数据清洗后放入数据仓库Hive中

对当天的数据url,channel,pv

[insert overwrite into url_channel select url,channel,count(1) pv from url_visit]

url获取。（类似于数据清洗）

获取以前从未抓过的url.

[select distinct url from url_channel t1 left outer join url_content t2 on t1.url=t2.url where t1.fetch_date=$day and t1.url is null]

形成一个文本文件（且用程序处理打乱排序）

Hive精准营销案例

抓取Url.[python的urlLib2,java的Boilerpipe]

抓完加载到hive中为后续处理准备,url_content表【url,title,body】

分词，为url打标签，赋值权重。

```
from( select a.url, a.channel、, a.pv, b.title, b.body from
url_channel a join url_content b on(a.url = b.url and
a.event_date=$date)) mapout
    insert overwrite table url_tag partition(event_date=$date)
reduce
mapout.url,mapout.channel,mapout.pv,mapout.title,mapout.body
using 'python url_info.py ${date}_total ' as url, products, contents,
attr, 3columns
```

Hive精准营销案例

以用户的维度合并行为数据。(uid为主键)

```
From (select uid_3, visittime, a.url, referer, useragent, channel,
keyword, area, b.products, b.contents, b.attr, b.3columns from
url_visit a LEFT OUTER join url_tag  b on(a.url = b.url and
b.fetch_date=$day) where a.fetch_date=$day and site in (  '0000',
'000', '000000')  distribute by uid_3 sort by uid_3,visittime) se
```

Hive精准营销案例

```
insert overwrite table user_action_day partition(fetch_date=$day)
    reduce
se.uid_3,se.visittime,se.url,se.referer,se.useragent,se.channel_id,
se.keyword, se.area,se.products, se.contents,se.attr,se.3columns
using 'python base_info.py ' as
uid,site,referer,time_du,agent,channel,area,keyword,attr,product,co
ntent,3columns ;
```

产生了用户的日行为。

Hive精准营销案例

汇总日行为数据，形成月度或时间区间行为数据。对上面的日数据进行处理，

```
CREATE  TABLE user_classify_center(
  uid string,   site string,
  referer string,   time_du string,
  agent string,   channel string,
  area string,   keyword string,
  attr string,   product string,
  content string,   3columns string)
PARTITIONED BY ( event_date int)
```

Hive精准营销案例

```
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.SequenceFileInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat'
LOCATION
  'hdfs://hangzhou-jishuan-
   MASTER01:9000/opt/bigdata/hive/warehouse/opt/bigdata_center.db/opt/bigdata_
   classify_center'
TBLPROPERTIES (
  'numPartitions'='113',
  'numFiles'='14370',
  'transient_lastDdlTime'='1383337550',
  'totalSize'='1558114192179',
  'numRows'='0',
  'rawDataSize'='0')
```

Hive精准营销案例

按照时间衰减函数对用户在各个时间段的分析，形成访问网站数据，访问频道数据，标签数据(喜好），标签数据（产品），访问时间数据（时段），地域数据等等。

有了这些数据就可以建立在上层的用户分群和精准营销了。

【电商可以发送更加精确的用户喜好产品；广告主可以对有相应爱好的区域发布广告。】

Hive精准营销案例

* 上层应用

----用户分群

行业（分析行业中产品关注度排名；购买行为分析；地域热度；关注的属性）

----网站

本网站的用户在整个网上有哪些行为。

访问某些Url的用户在网上有哪些行为。

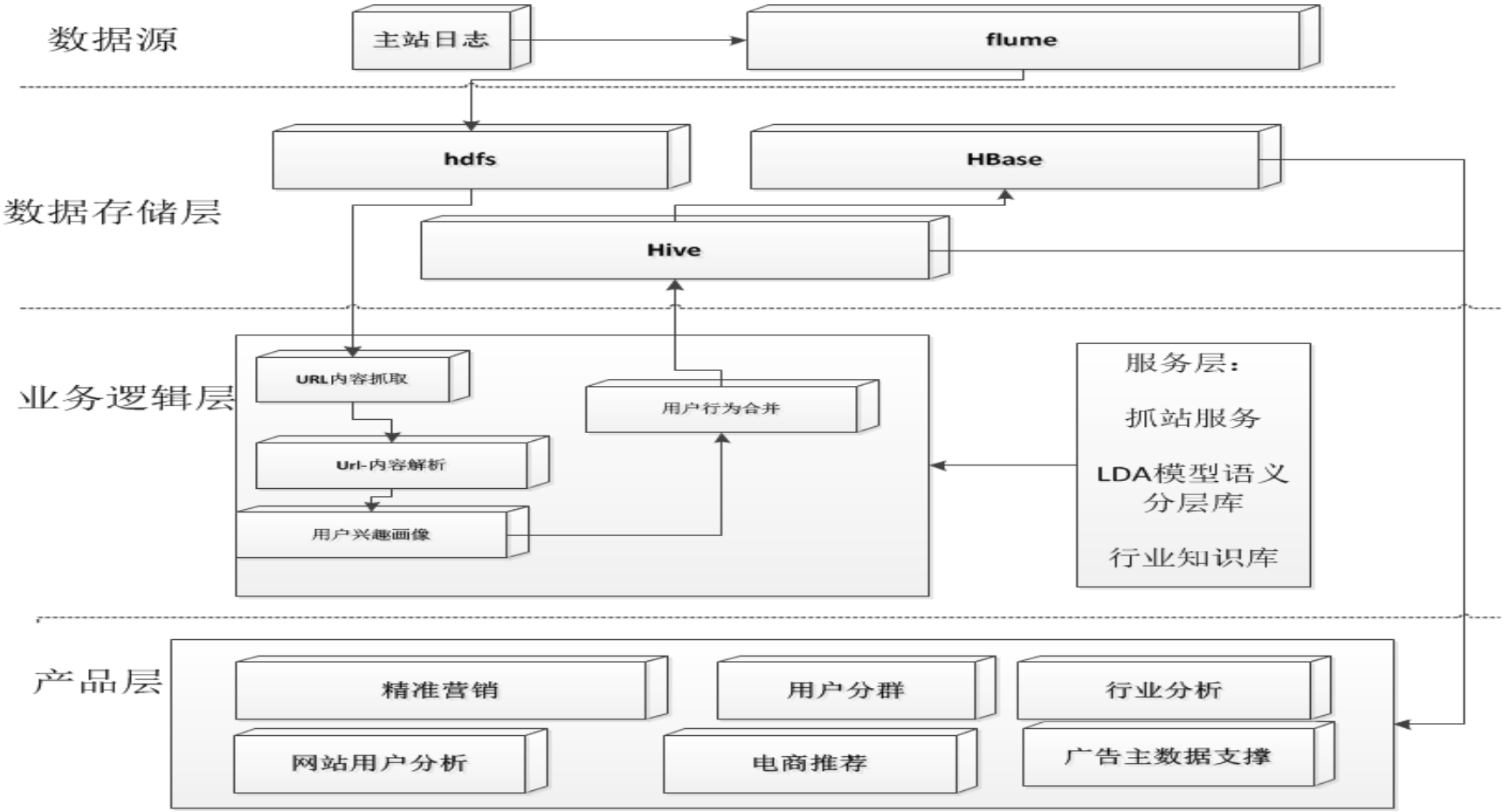
----精准营销

1.大批量数据交换，还是用数据分析

2.通过cookie实时查询相应用户的喜好。

放入hbase中，实时反馈向用户推荐。

Hive精准营销案例


数据源
主站日志
flume
数据存储层
hdfs
HBase
Hive
业务逻辑层
URL内容抓取
用户行为合并
Url-内容解析
用户兴趣画像
服务层：
抓站服务
LDA模型语义分层库
行业知识库
产品层
精准营销
用户分群
行业分析
网站用户分析
电商推荐
广告主数据支撑

上层应用的开发模式

Hive数据仓库

- 合并的用户行为数据

按相关维度对数据提取

- 根据具体应用统计或获取相关信息

分群数据或决策数据

- 将分析后的结果加载到数据库，供客户查看或供领导决策

小结

处理方式	优点	缺点	典型案例
自己写MapReduce任务	性能比Hive和Pig要高点	开发难度大	1） 搜索引擎网页处理，PageRank计算（Google）2） 典型的ETL（全盘扫描）3）机器学习/聚类，分类，推荐等（百度Ecomm）
使用Hive做基于SQL的分析	对于数据分析师来说SQL太熟悉了	有些场景下性能不如MR	1） 用户访问日志处理/互联网广告（Yahoo, Facebook, hulu, Amazon）2） 电子商务（淘宝的云梯）
使用Pig做数据分析	Pig的语法不是很普及	有些场景下性能不如MR	统计和机器学习（Yahoo, twitter）
基于HBase开发的系统	基本可以达到准实时统计分析功能	目前没有开源实现，开发成本高	大多是自有系统，例如Google的Percolator，淘宝的prom

谢谢！