数据仓库

<https://www.tuicool.com/wx/yEbm2en>

# 1 Hive入门

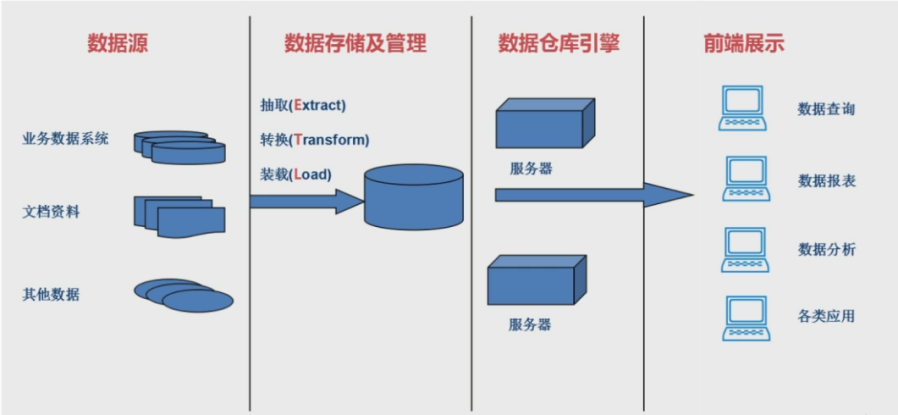
主要讲解hive入门的知识与实践

## 1.1 hive基本实践

### 1 数据仓库

数据仓库是一个面向主题的、集成的（来源多样：MySQL，Oracle，文本等）、定时更新（离线增量更新）的数据集，主要用途是查询。

#### 数据仓库的建立



#### OLTP与OLAP

OLTP -- 面向事物（要么成功，要么失败）：银行转账

OLTP是传统的关系型数据库的主要应用，主要是基本的、日常的事务处理，例如银行交易。

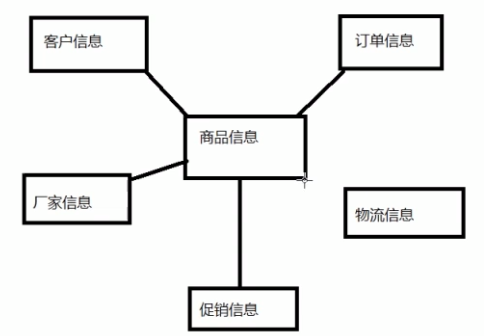
OLAP -- 联机分析处理（侧重于查询）：推荐系统

OLAP是数据仓库系统的主要应用，支持复杂的分析操作，侧重决策支持，并且提供直观易懂的查询结果。

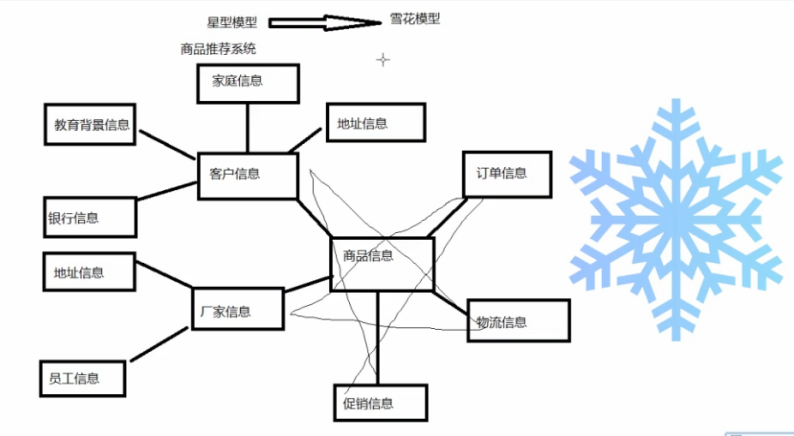
#### 数据仓库模型

星型模型和雪花模型

星型模型



雪花模型 -- 基于星型模型



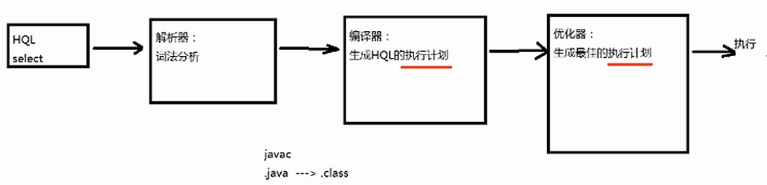
### 2 Hive安装实践

#### 1 hive元数据

hive将元数据存储在数据库中，支持mysql，derby，oracle等数据库。默认支持derby。

hive中的元数据包括表的名字，表的列和表的分区及其属性，表的属性包括（是否为外部表等）， 等。

#### 2 HQL的执行过程



编译器 🡪 类似于

javac: .java文件 🡪 .class文件

HQL 执行过程

解释器、编译器、优化器完成HQL查询语句从词法分析、语法分析、编译、优化以及查询计划（Plan）的生成。生成的查询计划存储在HDFS中，并在随后有MapReduce调用执行。

执行计划

执行计划（这里是查询计划！）

explain plan for select \* from emp where deptno10

查询执行计划（oracle中）

select \* from table(dbms\_xplan.display) ==> dbms\_xplan.display 固定写法!

全表扫描

全表扫描的效率会很低

创建索引

create index myindex on emp(deptno);

解释语句

explain plan for select \* from emp where deptno=10;

查看执行计划

select \* from table(dbms\_xplan.diisplay);

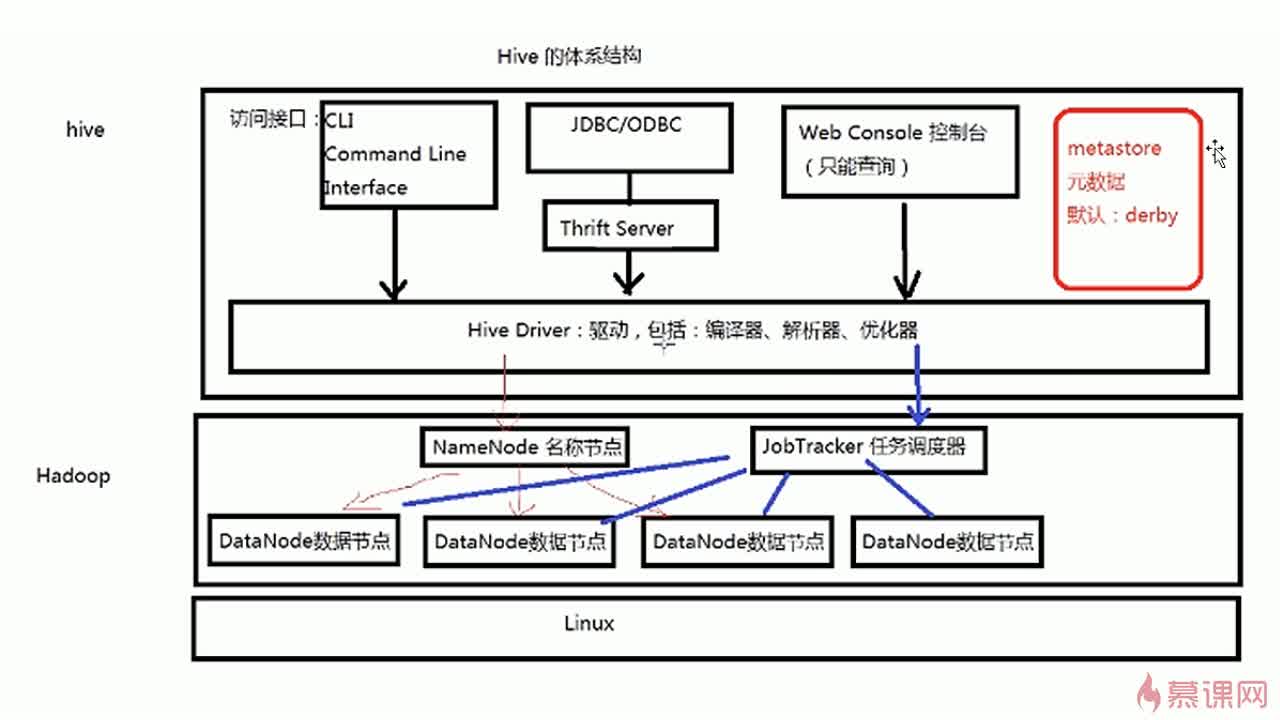
#### 3 Hive体系结构

## Hadoop

用HDFS进行存储，用Mapreduce进行计算

## 元数据存储（MetaStore）

通常是存储在关系型数据库如mysql，derby中



#### 4 Hive安装

三种模式

嵌入模式

元数据被存储在hive自带的derby数据库中

只允许创建一个连接（同一个时间只能有一个人在使用hive）

多用于demo

本地模式

元数据存储在MySQL数据库

MySQL和与hive运行在同一台机器上

多用于开发和测试

远程模式



配置环境变量

## HIVE\_HOME

export HIVE\_HOME=/home/hadoop/app/hive

export PATH=$PATH:$HIVE\_HOME/bin

嵌入模式

hive解压好后执行：

bin/hive

该命令会自动生成一个derby数据库metastore\_db，位于hive/metastore\_db下面

##### 本地模式安装

## 安装MySQL

$ sudo yum install mysql-server

$ sudo service mysqld start

$ sudo chkconfig mysqld on

设置密码：

安装的时候其实有提示的：

/usr/bin/mysqladmin -u root password 'new-password'

/usr/bin/mysqladmin -u root -h hadoop1 password 'new-password'

设置密码

$ mysqladmin -u root password "123456"

$ mysql -uroot -p

mysql存储元数据

首先在mysql创建hive数据库

mysql> create database hive;

配置hive-site.xml

在conf/目录下创建hive-site.xml

<?xml version="1.0" ?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>

<name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:mysql://hadoop1:3306/hive?createDatabaseIfNotExist=true&amp;useUnicode=true&amp;characterEncoding=UTF-8</value>

<description>JDBC connect string for a JDBC metastore</description>

</property>

<property>

<name>javax.jdo.option.ConnectionDriverName</name>

<value>com.mysql.jdbc.Driver</value>

</property>

<property>

<name>javax.jdo.option.ConnectionUserName</name>

<value>root</value>

</property>

<property>

<name>javax.jdo.option.ConnectionPassword</name>

<value>123456</value>

</property>

</configuration>

在lib 目录下放入mysql驱动器：

mysql-connector-java-5.1.44-bin.jar

启动hadoop

$ sbin/start-dfs.sh

mysql> GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'%' IDENTIFIED BY '123456' WITH GRANT OPTION;

mysql> flush privileges;

mysql> select user,host,password from user;

+------+-----------+-------------------------------------------+

| user | host | password |

+------+-----------+-------------------------------------------+

| root | localhost | \*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |

| root | hadoop1 | |

| root | 127.0.0.1 | |

| | localhost | |

| | hadoop1 | |

| root | % | \*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |

+------+-----------+-------------------------------------------+

mysql> delete from user where user='root' and host='hadoop1';

mysql> delete from user where user='root' and host='127.0.0.1';

依次，最后的结果是：

mysql> select user,host,password from user;

+------+------+-------------------------------------------+

| user | host | password |

+------+------+-------------------------------------------+

| root | % | \*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |

+------+------+-------------------------------------------+

符合要求

mysql> flush privileges; --刷新

启动hive

bin/hive

启动好以后，可以在mysql中

mysql> use hive;

mysql> show tables;

hive> create table test1(tid int,tname string);

hive> drop table test1;

hive实例：



create table course

(num int,course string)

row format delimited fields terminated by ',';

load data local inpath '/home/hadoop/app/hive/files/courseexample/course.txt'

into table course;

create table student

(id int , name string , sex string ,age int , deparment string)

row format delimited fields terminated by ',';

load data local inpath "/home/hadoop/app/hive/files/courseexample/student.txt"

into table student;

create table studentcourse

(id int,num int,score int)

row format delimited fields terminated by ',';

load data local inpath '/home/hadoop/app/hive/files/courseexample/studentcourse.txt'

into table studentcourse;

### 3 hive管理

hive启动方式：

Cli（命令行）

web界面方式

远程服务启动方式

#### 1 Cli命令行

bin/hive

或者

$ hive --service cli

【备注】启动报错 Name node is in safe mode.

namenode在安全模式下

【解决方案】

bin/hadoop dfsadmin -safemode leave

!!!注意

用户可以通过dfsadmin -safemode value 来操作安全模式，参数value的说明如下：

enter - 进入安全模式

leave - 强制NameNode离开安全模式

get - 返回安全模式是否开启的信息

wait - 等待，一直到安全模式结束。

【可能原因】hadoop关闭停止不规范，最好关机前手动关闭hadoop服务

关闭：

hive> quit;

hive> exit;

查询语句

1. hive命令行直接查询
2. source命令查询

hive> source /home/hadoop/app/hive/files/courseexample/course\_test.sql;

3)hive -e 查询

$ hive -e 'show tables'

2 web UI

需要做一定的配置

1. 下载hive src源码
2. 解压打包

jar cvfM0 hive-hwi-1.1.0.war -C web/ .

拷贝到hive/lib目录

cp hive-hwi-1.1.0.war /home/hadoop/app/hive/lib/

1. 修改配置文件conf/hive-site.xml

<property>

<name>hive.hwi.listen.host</name>

<value>0.0.0.0</value>

<description>This is the host address the Hive Web Interface will listen on</description>

</property>

<property>

<name>hive.hwi.listen.port</name>

<value>9999</value>

<description>This is the port the Hive Web Interface will listen on</description>

</property>

<property>

<name>hive.hwi.war.file</name>

<value>/lib/hive-hwi-1.1.0.war</value>

<description>This is the WAR file with the jsp content for Hive Web Interface</description>

</property>

1. 拷贝jdk/lib/tool.jar到hive/lib

cp /home/hadoop/app/jdk/lib/tools.jar /home/hadoop/app/hive/lib/

1. 登陆网页查询

<http://hadoop1:9999/hwi/>

1. web只支持查询功能

### 4 hive数据存储

基于HDFS

没有专门的数据存储格式

存储格式主要包括：数据库、文件、表、视图

创建表时，指定hive数据的列分隔符与行分隔符

表:

内部表 Table

分区表 Partition Table

外部表 External Table

桶表 Bucket Table

视图

#### 1 内部表

与数据库中table概念是类似的

每一个table在hive中都有一个相应的目录存储数据

所有的table数据（不包括external table）都存储在该目录中

删除表时，元数据与数据都会被删除

create table t3(

id int,

name string,

age int

)

row format delimited fields terminated by ',' --指定行分隔符和列分隔符，行分隔符默认是制表符

location '/user/hive/warehouse/mytable/t3'

;

#### 2 分区表

partition对应于数据库中的partition列的密集索引

在hive中，表中的一个partition对应于表下的一个目录，所有的partition的数据都存储在对应的目录中。

create table partition\_table\_1(

sid int,

sname string)

partitioned by(gender string)

row format delimited fields terminated by ',';

查询时，如果查询性别为男性，只需要到指定分区目录查询，提高查询效率

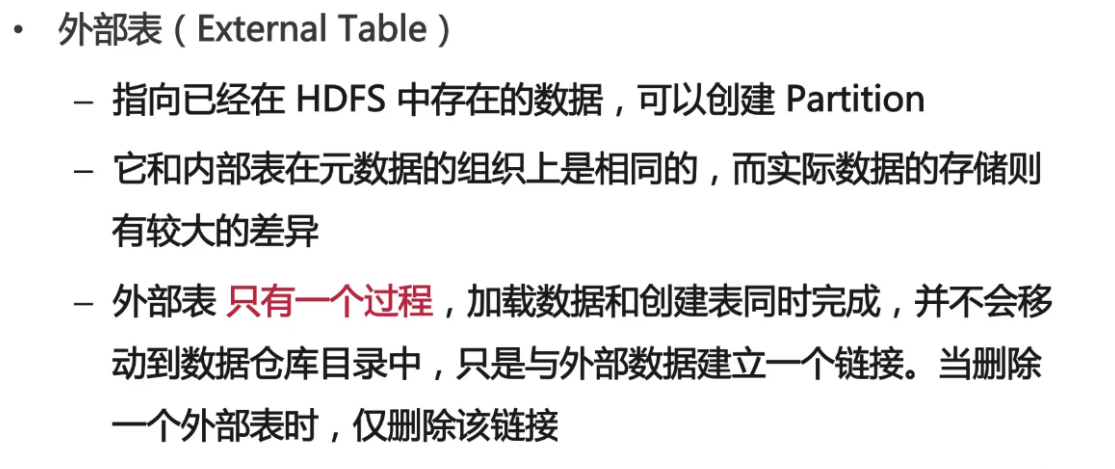
insert into table partition\_table partition(gender='M')

select sid,sname from sample\_data where gender='M'

insert into table partition\_table partition(gender='F')

select sid,sname from sample\_data where gender='F'

3 外部表



指向已经再HDFS中存在的数据，可以创建Partition

它和内部表在元数据组织上是相同的；而实际数据的存储则存在较大的差异。

外部表只是一个过程，加载数据和创建表同时完成，并不会一定到数据仓库目录中，只是与外部数据建立一个链接。当删除一个外部表时，仅删除该链接

#### 4 桶表(Bucket Table)

桶表是对数据进行哈希取值，然后放到不同文件存储。也就是说，桶表中的数据，是通过哈希运算后，将其打散，再存入文件当中，这样做会避免造成热块，从而提高查询速度。

create table bucket\_table1 (name string,age int)

clustered by(name)into 5 buckets;

如果使用一些Hash算法（比如MD5(tom)=34b7da764b21d298ef307d04d8152dc5），都变成一样长度的字符，这样0到F十六种字符开头的概率是一样的，分16个桶基本平均分配了，分2、4、6、8个桶也很简单。

#### 5 视图

视图(View)

视图是一种虚表，是一个逻辑概念；可以跨越多张表

-视图建立在已有基础上，视图赖以建立的这些表称为基表

-视图可以简化复杂的查询

- 视图不存数据

# 面试准备

## 1.2 hive调优

参考文档

https://www.cnblogs.com/smartloli/p/4356660.html

### 1 配置角度优化

#### 列裁剪

Hive 在读数据的时候，可以只读取查询中所需要用到的列，而忽略其它列。

SELECT a,b FROM q WHERE e<10;

在实施此项查询中，Q 表有 5 列（a，b，c，d，e），Hive 只读取查询逻辑中真实需要 的 3 列 a、b、e，而忽略列 c，d；这样做节省了读取开销，中间表存储开销和数据整合开销。

set hive.optimize.cp=true;## （默认值为真）

#### 分区裁剪

SELECT \* FROM (SELECTT a1,COUNT(**1**) FROM T GROUP BY a1) subq WHERE subq.prtn=**100**; #（多余分区）

SELECT \* FROM T1 JOIN (SELECT \* FROM T2) subq ON (T1.a1=subq.a2) WHERE subq.prtn=**100**;

查询语句若将“subq.prtn=100”条件放入子查询中更为高效，可以减少读入的分区 数目。 Hive 自动执行这种裁剪优化。

分区参数为：

set hive.optimize.pruner=true;##（默认值为真）

#### join操作

在编写带有 join 操作的代码语句时，应该**将条目少的表/子查询放在 Join 操作符的左边**。 因为在 Reduce 阶段，位于 Join 操作符左边的表的内容会被加载进内存，载入条目较少的表 可以有效减少 OOM（out of memory）即内存溢出。所以对于同一个 key 来说，对应的 value 值小的放前，大的放后，这便是“**小表放前**”原则。 若一条语句中有多个 Join，依据 Join 的条件相同与否，有不同的处理方法。

**Join原则**

1. join条件相同

如果 Join 的 key 相同，不管有多少个表，都会则会合并为一个 Map-Reduce

INSERT OVERWRITE TABLE pv\_users

SELECT pv.pageid, u.age FROM page\_view p

JOIN user u ON (pv.userid = u.userid)

JOIN newuser x ON (u.userid = x.userid);

一个 Map-Reduce 任务，而不是 ‘n’ 个

1. join条件不同

Map-Reduce 的任务数目和 Join 操作的数目是对应的。

INSERT OVERWRITE TABLE pv\_users

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);

此时，mapreduce任务数应该是2个。

#### Map join

Join 操作在 Map 阶段完成，不再需要Reduce，前提条件是需要的数据在 Map 的过程中可以访问到。比如查询：

这是在0.7.0之前的做法

INSERT OVERWRITE TABLE pv\_users

SELECT /\*+ MAPJOIN(pv) \*/ pv.pageid, u.age

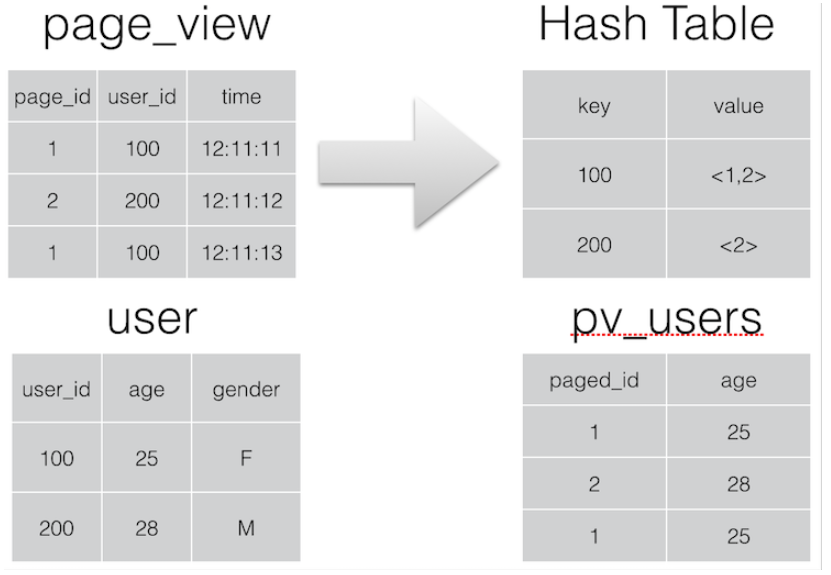
FROM page\_view pv

JOIN user u ON (pv.userid = u.userid);

在hive高级版本中，只需要配置参数：

set hive.auto.convert.join=true;

可以在 Map 阶段完成 Join，如图所示：



相关的参数为：

hive.join.emit.interval = 1000

hive.mapjoin.size.key = 10000

hive.mapjoin.cache.numrows = 10000

#### Group by操作

进行GROUP BY操作时需要注意一下几点：

* Map端部分聚合

事实上并不是所有的聚合操作都需要在reduce部分进行，很多聚合操作都可以先在Map端进行部分聚合，然后reduce端得出最终结果。

这里需要修改的参数为：

　　hive.map.aggr=true（用于设定是否在 map 端进行聚合，默认值为真） hive.groupby.mapaggr.checkinterval=100000（用于设定 map 端进行聚合操作的条目数）

* 有数据倾斜时进行负载均衡

此处需要设定 hive.groupby.skewindata，当选项设定为 true 是，生成的查询计划有两个 MapReduce任务。在第一个 MapReduce 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 能分发到不同的 reduce 中，从而达到负载均衡的目的；第二个 MapReduce 任务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 分布到同一个 reduce 中），最后完成最终的聚合操作。

set hive.groupby.skewindata=true;

#### 合并小文件

我们知道文件数目小，容易在文件存储端造成瓶颈，给 HDFS 带来压力，影响处理效率。对此，可以通过合并Map和Reduce的结果文件来消除这样的影响。

用于设置合并属性的参数有：

* 是否合并Map输出文件：hive.merge.mapfiles=true（默认值为真）
* 是否合并Reduce 端输出文件：hive.merge.mapredfiles=false（默认值为假）
* 合并文件的大小：hive.merge.size.per.task=256\*1000\*1000（默认值为 256000000）

### 2程序角度优化

#### 无效ID在关联时的数据倾斜问题

问题：日志中常会出现信息丢失，比如每日约为 20 亿的全网日志，其中的 user\_id 为主 键，在日志收集过程中会丢失，出现主键为 null 的情况，如果取其中的 user\_id 和 bmw\_users 关联，就会碰到数据倾斜的问题。原因是 Hive 中，主键为 null 值的项会被当做相同的 Key 而分配进同一个计算 Map。

解决方法 1：user\_id 为空的不参与关联，子查询过滤 null

SELECT \* FROM log a

JOIN bmw\_users b ON a.user\_id IS NOT NULL AND a.user\_id=b.user\_id

UNION All SELECT \* FROM log a WHERE a.user\_id IS NULL

解决方法 2 如下所示：函数过滤 null

SELECT \* FROM log a

LEFT OUTER JOIN bmw\_users b

ON CASE WHEN a.user\_id IS NULL THEN CONCAT(‘dp\_hive’,RAND()) ELSE a.user\_id END =b.user\_id;

-- 此处，把空值的 key 变成一个字符串加上随机数，就能把倾斜的 数据分到不同的Reduce上，从而解决数据倾斜问题。因为空值不参与关联，即使分到不同 的 Reduce 上，也不会影响最终的结果。

调优结果：原先由于数据倾斜导致运行时长超过 1 小时，解决方法 1 运行每日平均时长 25 分钟，解决方法 2 运行的每日平均时长在 20 分钟左右。优化效果很明显。

我们在工作中总结出：解决方法2比解决方法1效果更好，不但IO少了，而且作业数也少了。解决方法1中log读取两次，job 数为2。解决方法2中 job 数是1。这个优化适合无效 id（比如-99、 ‘’，null 等）产生的倾斜问题。

#### 不同数据类型关联产生的倾斜问题

场景：用户表中user\_id为int，log表中的user\_id为int和string。当按照user\_id进行两个表的join操作的时候，默认的hash操作会按照int类型的id进行分配，这样导致所有的string类型的数据都进入同一个reduce之中，解决的方法如下，把所有的数字类型转换为string的类型：

select \* from users a

left outer join logs b

on a.usr\_id = cast(b.user\_id as string)

【备注】

left join , right join, inner Join,outer join区别

join –-> inner join ##(inner join返回两张表满足on条件的数据)

left join 等价于 left outer join

#### count distinct优化

（不存在数据倾斜—---优化而已）

数据量特别大的时候，使用distinct去重容易导致数据倾斜

例如 count(distinct ip)

可以用group by来优化

**select count(\*) from group by ip**

### 3 数据倾斜优化

#### join产生数据倾斜

**## 大表和小表join**

**产生原因**：Hive在进行join时，按照join的key进行分发，而在join左边的表的数据会首先读入内存，如果左边表的key相对分散，读入内存的数据会比较小，join任务执行会比较快；而如果左边的表key比较集中，而这张表的数据量很大，那么数据倾斜就会比较严重，而如果这张表是小表，则还是应该把这张表放在join左边。  
 **解决方式**：使用map join让小的维度表先进内存。在map端完成reduce。Left join时，小表放左边，参数配置：

set hive.auto.convert.join=true;

## 大表和大表join

无效ID在关联时的数据倾斜问题

不同数据类型关联产生的倾斜问题

### 4 索引

#### SQL 索引

目的：提高Hive表指定列的查询速度

SQL索引有两种，聚集索引和非聚集索引。

举例：

字典查字。

按拼音索引 —-> 聚集索引

按偏旁索引 --> 非聚集索引

**原理**

聚集索引存储记录是物理上连续存在，而非聚集索引是逻辑上的连续，物理存储并不连续。聚集索引和非聚集索引的根本区别是表记录的排列顺序和与索引的排列顺序是否一致。

举字典的例子：如果按照拼音查询，那么都是从a-z的，是具有连续性的，a后面就是b，b后面就是c， 聚集索引就是这样的，他是和表的物理排列顺序是一样的，例如有id为聚集索引，那么1后面肯定是2,2后面肯定是3，所以说这样的搜索顺序的就是聚集索引。非聚集索引就和按照部首查询是一样是，可能按照偏房查询的时候，根据偏旁‘弓’字旁，索引出两个汉字，张和弘，但是这两个其实一个在100页，一个在1000页，（这里只是举个例子），他们的索引顺序和数据库表的排列顺序是不一样的，这个样的就是非聚集索引。

**存储**

聚集索引就是在数据库中开辟一个物理空间存放他的排列的值，例如1-100，所以当插入数据时，他会重新排列整个整个物理空间，而非聚集索引其实可以看作是一个含有聚集索引的表，他只仅包含原表中非聚集索引的列和指向实际物理表的指针。

建立索引原则

#### 建立索引的原则

1) 定义主键的数据列一定要建立索引。

2) 定义有外键的数据列一定要建立索引。

3) 对于经常查询的数据列最好建立索引。

#### 索引分类

mysql的索引分为单列索引(主键索引,唯索引,普通索引)和组合索引.

单列索引:一个索引只包含一个列,一个表可以有多个单列索引.

组合索引:一个组合索引包含两个或两个以上的列,

CREATE TABLE `award` (

`id` int(**11**) NOT NULL AUTO\_INCREMENT COMMENT '用户id',

`aty\_id` varchar(**100**) NOT NULL DEFAULT '' COMMENT '活动场景id',

`nickname` varchar(12) NOT NULL DEFAULT '' COMMENT '用户昵称',

`is\_awarded` tinyint(**1**) NOT NULL DEFAULT 0 COMMENT '用户是否领奖',

`award\_time` int(**11**) NOT NULL DEFAULT 0 COMMENT '领奖时间',

`account` varchar(**12**) NOT NULL DEFAULT '' COMMENT '帐号',

`password` char(32) NOT NULL DEFAULT '' COMMENT '密码',

`message` varchar(**255**) NOT NULL DEFAULT '' COMMENT '获奖信息',

`created\_time` int(**11**) NOT NULL DEFAULT 0 COMMENT '创建时间',

`updated\_time` int(**11**) NOT NULL DEFAULT 0 COMMENT '更新时间',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=**1** DEFAULT CHARSET=utf8 COMMENT='获奖信息表';

创建索引

## 单列索引

# 方法1

CREATE INDEX account\_Index ON `award`(`account`); -- award是表名，account是字段名

# 方法2

ALTER TABLE award ADD INDEX account\_Index(`account`)

-----------------------------------------------------------------

 CREATE INDEX IndexName ON `TableName`(`字段名`(length))

ALTER TABLE TableName ADD INDEX IndexName(`字段名`(length))