Hive基础总结(面试常用)

hive是基于Hadoop的一个数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供简单的sql查询功能，可以将sql语句转换为MapReduce任务进行运行。 Metastore （hive元数据） Hive将元数据存储在数据库中，比如mysql ,derby.Hive中的元数据包括表的名称，表的列和分区及其属性，表的数据所在的目录 Hive数据存储在HDFS，大部分的查询、计算由mapreduce完成 Hive数据仓库于数据库的异同

（1）由于Hive采用了SQL的查询语言HQL，因此很容易将Hive理解为数据库。其实从结构上来看，Hive和数据库除了拥有类似的查询语言， 再无类似之处。

（2）数据存储位置。 hdfs raw local fs

（3）数据格式。 分隔符

（4）数据更新。hive读多写少。Hive中不支持对数据的改写和添加，所有的数据都是在加载的时候中确定好的。 INSERT INTO … VALUES添加数据，使用UPDATE … SET修改数据 不支持的 HDFS 一次写入多 次读取
（5） 执行。hive通过MapReduce来实现的 而数据库通常有自己的执行引擎。

（6）执行延迟。由于没有索引，需要扫描整个表，因此延迟较高。另外一个导致Hive执行延迟高的因素是MapReduce框架

（7）可扩展性

（8）数据规模。 hive几种基本表类型：内部表、外部表、分区表、桶表 内部表（管理表）和外部表的区别： 创建表 外部表创建表的时候，不会移动数到数据仓库目录中（/user/hive/warehouse），只会记录表数据存放的路径 内部表会把数据复制或剪切到表的目录下 删除表 外部表在删除表的时候只会删除表的元数据信息不会删除表数据 内部表删除时会将元数据信息和表数据同时删除 表类型一、管理表或内部表Table Type: MANAGED_TABLE

```
create table if not exists dept(
deptno int,
deptname string,
address string
)
row  format delimited fields terminated by '\t';

//加载HDFS文件到Hive表中
load data inpath '/input/dept.txt' into table dept;

//用来指定原文件的列分隔符
row  format delimited fields terminated by '\t';
load 如果操作的HDFS上的文件，代表着会移动或者剪切文件

desc formatted dept;  //描述表结构信息
Location:               hdfs://bigdata/user/hive/warehouse/db01.db/dept
Table Type:             MANAGED_TABLE
```

表类型二、外部表

```
create  external table emp(
empno int,
empname string,
```

```
empjob string,
mgno int,
birthday string,
salary float,
bonus float,
depno int
)
row  format delimited fields terminated by '\t'
location '/input/demo';

//描述表结构
desc formatted emp;
Location:               hdfs://bigdata/input/demo
Table Type:             EXTERNAL_TABLE

删除内部表
drop table dept;

删除外部表
drop table emp;

清空表数据
truncate table student;
```

表类型三、分区表 分区表创建表的时候需要指定分区字段，分区字段与普通字段的区别：分区字段会在HDFS表目录下生成一个分区字段名称的目录，而普通字段则不会，查询的时候可以当成普通字段来使用，一般不直接和业务直接相关。

```
create  table emp_part(
 empno int,
 empname string,
 empjob string,
 mgrno int,
 birthday string,
 salary float,
 bonus float,
 deptno   int
 )
 partitioned by (province string)
row format delimited fields terminated by '\t';

//向分区表加载数据
load data local inpath '/home/user01/emp.txt' into table emp_part partition
(province='CHICAGO');

//描述表信息
desc formatted  emp_part;

//查询全表数据
select * from emp_part;

//查询分区字段表数据
select * from emp_part where province='CHICAGO';

//查看分区信息
show partitions emp_part;

//增加分区
aler table emp_part add [if not exist] partition(provine='zhejiang',city='hangzhou')
```

```
//删除分区
aler table emp_part drop [if  exist] partition(provine='zhejiang',city='hangzhou')
```

外部分区表

```
create external  table  dept_part(
deptno int,
deptname string,
address string
)
partitioned by (province string)
row  format delimited fields terminated by '\t'
location '/input/demo';

//手动增加分区字段及外部目录:
alter table dept_part add partition (province='BOSTON') location '/input/demo/BOSTON';

//手动增加分区字段（自动生成分区目录）
alter table dept_part add partition (province='NEW YORK');
```

表类型四：桶表 将内部表，外部表和分区表进一步组织成桶表 可以将表的列通过Hash算法进一步分解成不同的文件存储

```
create table test_bucket_table(
id int,
name string
)
clustered by (id) into 5 bucket;
```

创建表的方式 方式一 create + load

```
create [external] table table_name(
    col1_name col1_type,
    ...
    coln_name coln_type
    )
    row  format delimited fields terminated  by '\t';

   //load加载数据
    laod data [local] inpth '本地文件(linux)/HDFS' [overwrite] into table  table_name;
```

方式二 like + load

```
//复制表结构
    create table tableB like tableA;    //首先必须要有tableA

//load加载数据
    laod data [local] inpth '本地文件(linux)/HDFS' [overwrite] into table  table_name;
```

方式三 as 创建表的同时加载数据

```
create table emp_insert(
id int,
name string,
job string,
salary float
)
row format  delimited fields terminated by ',';

//insert into 加载数据
insert into table emp_insert select empno,empname,empjob,salary from emp_part1 where
day='20170308' and  hour='14';
```

方式四 create + insert //创建表

```
create table emp_insert(
id int,
name string,
job string,
salary float
)
row format  delimited fields terminated by ',';

//insert into 加载数据
insert into table emp_insert select empno,empname,empjob,salary from emp_part1 where
day='20170308' and  hour='14';
```

加载数据的方式 加载方式一

```
//加载本地文件到Hive表    --使用存储介质（移动硬盘）
laod data local inpth '本地文件(linux)' [overwrite] into table  table_name;
```

加载方式二

```
//创建表时通过select查询语句加载数据
 create table tableB row format delimited filelds termianted by ','  as select * from tableA;
```

加载方式三

```
//创建表时通过select查询语句加载数据
 create table tableB row format delimited filelds termianted by ','  as select * from tableA;
```

加载方式四

```
//创建表时通过select查询语句加载数据
 create table tableB row format delimited filelds termianted by ','  as select * from tableA;
```

加载方式五

```
//先创建表，通过insert into table table_namea  select  *  fom tableB
```

加载方式六

```
<property>
  <name>hive.fetch.task.conversion</name>
  <value>more</value>
  <description>
    Some select queries can be converted to single FETCH task
    minimizing latency.Currently the query should be single
    sourced not having any subquery and should not have
    any aggregations or distincts (which incurrs RS),
    lateral views and joins.
    1. minimal : SELECT STAR, FILTER on partition columns, LIMIT only
    2. more    : SELECT, FILTER, LIMIT only (+TABLESAMPLE, virtual columns)
  </description>
</property>
```

几种导出数据的方式

1.insert overwrite ... 导出到本地目录

```
insert overwrite local directory '/home/user01/export' row format delimited fields terminated by
' ' select * from emp_part1;
```

2.insert overwrite ... 导出到HDFS之上

```
insert overwrite directory '/export' select * from emp_part1 where day='20170308';
```

3.hive -e 'HQL query' >

```
test    bin/hive -e 'select * from db01.student' >> test.txt
```

4)sqoop Hive 自定义函数函数 UDF 一进一出 处理原文件内容某些字段包含 [] “” UDAF 多进一出 sum() avg() max() min() UDTF 一进多出 ip -> 国家 省 市 Hive4种排序

order by //可以指定desc 降序 asc 升序

order by会对输入做全局排序，因此只有一个Reducer(多个Reducer无法保证全局有序)，然而只有一个Reducer，会导致当输入规模较大时，消耗较长的计算时间。

sort by 【对分区内的数据进行排序】

sort by不是全局排序，其在数据进入reducer前完成排序，因此，如果用sort by进行排序，并且设置mapred.reduce.tasks>1，则sort by只会保证每个reducer的输出有序，并不保证全局有序。

sort by不同于order by，它不受Hive.mapred.mode属性的影响，sort by的数据只能保证在同一个reduce中的数据可以按指定字段排序。使用sort by你可以指定执行的reduce个数(通过set mapred.reduce.tasks=n来指定)，对输出的数据再执行归并排序，即可得到全部结果。

distribute by 【对map输出进行分区】

distribute by是控制在map端如何拆分数据给reduce端的。hive会根据distribute by后面列，对应reduce的个数进行分发，默认是采用hash算法。sort by为每个reduce产生一个排序文件。在有些情况下，你需要控制某个特定行应该到哪个reducer，这通常是为了进行后续的聚集操作。

distribute by刚好可以做这件事。因此，distribute by经常和sort by配合使用。

cluster by　cluster by除了具有distribute by的功能外还兼具sort by的功能。当distribute by和sort by 是同一个字段的时候可以使用cluster by替代。但是排序只能是倒叙排序，不能指定排序规则为ASC或者DESC。 三种分组的区别

row_number：不管col2字段的值是否相等，行号一直递增，比如：有两条记录的值相等，但一个是第一，一个是第二　rank：上下两条记录的col2相等时，记录的行号是一样的，但下一个col2值的行号递增N（N是重复的次数），比如：有两条并列第一，下一个是第三，没有第二　dense_rank：上下两条记录的col2相等时，下一个col2值的行号递增1，比如：有两条并列第一，下一个是第二 Hive优化

1.fetch task任务不走MapReduce，可以在hive配置文件中设置最大化和最小化fetch task任务；通常在使用hiveserver2时调整为more；

设置参数的优先级：

在命令行或者代码设置参数

```
hive-site.xml>hive-default.xml set hive.fetch.task.conversion=more;
```

单次交互模式下有效， bin/hive --hiveconf hive.fetch.task.conversion=more 上面的两种方法都可以开启了Fetch任务，但是都是临时起作用的；如果你想一直启用这个功能，可以在${HIVE_HOME}/conf/hive-site.xml里面加入以下配置：

```
<property>
  <name>hive.fetch.task.conversion</name>
  <value>more</value>
  <description>
    Some select queries can be converted to single FETCH task
    minimizing latency.Currently the query should be single
    sourced not having any subquery and should not have
    any aggregations or distincts (which incurrs RS),
    lateral views and joins.
    1. minimal : SELECT STAR, FILTER on partition columns, LIMIT only
    2. more    : SELECT, FILTER, LIMIT only (+TABLESAMPLE, virtual columns)
  </description>
</property>
```

2.strict mode：严格模式设置，严格模式下将会限制一些查询操作　文件格式，ORC PARQUET 等　分区表　select查询不加where过滤条件，不会执行 开启严格模式 hive提供的严格模式，禁止3种情况下的查询模式。 a：当表为分区表时，where字句后没有分区字段和限制时，不允许执行。 b：当使用order by语句时，必须使用limit字段，因为order by 只会产生一个reduce任务。 c：限制笛卡尔积的查询。sql语句不加where不会执行

```
<property>
  <name>hive.mapred.mode</name>
  <value>nonstrict</value>
  <description>The mode in which the Hive operations are being performed.
     In strict mode, some risky queries are not allowed to run. They include:
       Cartesian Product.
       No partition being picked up for a query.
       Comparing bigints and strings.
       Comparing bigints and doubles.
       Orderby without limit.
  </description>
</property>
```

3.优化sql语句，如先过滤再join，先分组再做distinct;

```
Select count(*) cnt
From store_sales ss
     join household_demographics hd on (ss.ss_hdemo_sk = hd.hd_demo_sk)
     join time_dim t on (ss.ss_sold_time_sk = t.t_time_sk)
     join store s on (s.s_store_sk = ss.ss_store_sk)
Where
     t.t_hour = 8
     t.t_minute >= 30
     hd.hd_dep_count = 2
order by cnt;
```

4.MapReduce过程的map、shuffle、reduce端的snappy压缩　需要先替换hadoop的native本地包开启压缩　在mapred-site.xml文件设置启用压缩及压缩编码　在执行SQL执行时设置启用压缩和指定压缩编码

```
set mapreduce.output.fileoutputformat.compress=true;
set mapreduce.output.fileoutputformat.compress.codec=org apache.hadoop.io.compress.SnappyCodec;
```

5.大表拆分成子表，提取中间结果集，减少每次加载数据　多维度分析，多个分析模块　每个分析模块涉及字段不一样，而且并不是表的全部字段

6.分区表及外部表　设计二级分区表（一级字段为天，二级字段设置小时）　创建的的是外部表，创建表时直接指定数据所在目录即可，不用再用load加载数据

7.设置map和reduce个数：默认情况下一个块对应一个map任务，map数据我们一般不去调整，reduce个数根据reduce处理的数据量大小进行适当调整体现“分而治之”的思想

```
hive-site.xml
hive.mapred.reduce.tasks.speculative.execution=true;
<property>
  <name>hive.mapred.reduce.tasks.speculative.execution</name>
  <value>true</value>
  <description>Whether speculative execution for reducers should be turned on. </description>
</property>
```

8.JVM重用：一个job可能有多个map reduce任务，每个任务会开启一个JVM虚拟机，默认情况下一个任务对应一个JVM，任务运行完JVM即销毁，我们可以设置JVM重用参数，一般不超过5个，这样一个JVM内可以连续运行多个任务 JVM重用是Hadoop调优参数的内容，对Hive的性能具有非常大的影响，特别是对于很难避免小文件的场景或者task特别多的场景，这类场景大多数执行时间都很短。hadoop默认配置是使用派生JVM来执行map和reduce任务的，这是jvm的启动过程可能会造成相当大的开销，尤其是执行的job包含有成千上万个task任务的情况。 JVM重用可以使得JVM实例在同一个JOB中重新使用N次，N的值可以在Hadoop的mapre-site.xml文件中进行设置（建议参考5~10）

mapred.job.reuse.jvm.num.tasks(旧版)

mapreduce.job.jvm.numtasks(新版)

hadoop.apache.org/docs/r2.5.2/hadoop-mapreduce-client/hadoop-mapreduce-client-core/mapred-default.xml http://hadoop.apache.org/docs/r2.5.2/hadoop-mapreduce-client/hadoop-mapreduce-client-core/mapred-default.xml 也可在hive的执行设置：

```
hive-site.xml
hive.mapred.reduce.tasks.speculative.execution=true;
<property>
  <name>hive.mapred.reduce.tasks.speculative.execution</name>
  <value>true</value>
  <description>Whether speculative execution for reducers should be turned on. </description>
</property>
```

9.推测执行：例如一个Job应用有10个MapReduce任务（map 及reduce），其中9个任务已经完成，那么application Master会在另外启动一个相同的任务来运行未完成的那个，最后哪个先运行完成就把另一个kill掉　启用speculative最大的好处是，一个map执行的时候，系统会在其他空闲的服务器上启动相同的map来同时运行，哪个运行的快就使用哪个的结果，另一个运行慢的在有了结果之后就会被kill。

```
hive-site.xml
hive.mapred.reduce.tasks.speculative.execution=true;
<property>
  <name>hive.mapred.reduce.tasks.speculative.execution</name>
  <value>true</value>
  <description>Whether speculative execution for reducers should be turned on. </description>
</property>
```

数据倾斜 对于普通的join操作，会在map端根据key的hash值，shuffle到某一个reduce上去，在reduce端做join连接操作，内存中缓存join左边的表，遍历右边的表，依次做join操作。所以在做join操作时候，将数据量多的表放在join的右边。

当数据量比较大，并且key分布不均匀，大量的key都shuffle到一个reduce上了，就出现了数据的倾斜。 常见的数据倾斜出现在group by和join..on..语句中。

join（数据倾斜） 在进行两个表join的过程中，由于hive都是从左向右执行，要注意讲小表在前，大表在后（小表会先进行缓存）。

map/reduce程序执行时，reduce节点大部分执行完毕，但是有一个或者几个reduce节点运行很慢，导致整个程序的处理时间很长，这是因为某一个key的条数比其他key多很多（有时是百倍或者千倍之多），这条key所在的reduce节点所处理的数据量比其他节点就大很多，从而导致某几个节点迟迟运行不完，此称之为数据倾斜。

hive在跑数据时经常会出现数据倾斜的情况，使的作业经常reduce完成在99%后一直卡住，最后的１%花了几个小时都没跑完，这种情况就很可能是数据倾斜的原因，

```
hive.groupby.skewindata=true;
```

如果是group by过程出现倾斜应将此项设置true。

```
<property>
    <name>hive.groupby.skewindata</name>
    <value>false</value>
    <description>Whether there is skew in data to optimize group by queries</description>
</property>
```

```
hive.optimize.skewjoin.compiletime=true;
```

如果是join 过程中出现倾斜应将此项设置为true 不影响结果可以考虑过滤空值

```
<property>
    <name>hive.optimize.skewjoin.compiletime</name>
    <value>false</value>
</property>
```

```
 hive.optimize.skewjoin.compiletime=true;
```

如果是join过程出现倾斜应该设置为true 此时会将join语句转化为两个mapreduce任务，第一个会给jion字段加随机散列 set hive.skewjoin.key=100000;

这个是join的键对应的记录条数超过这个值则会进行优化。

可以在空值前面加随机散列 3种常见的join Map-side Join mapJoin的主要意思就是，当链接的两个表是一个比较小的表和一个特别大的表的时候，我们把比较小的table直接放到内存中去，然后再对比较大的表格进行map操作。

join就发生在map操作的时候，每当扫描一个大的table中的数据，就要去去查看小表的数据，哪条与之相符，继而进行连接。这里的join并不会涉及reduce操作。map端join的优势就是在于没有shuffle，真好。

在实际的应用中，我们这样设置：

set hive.auto.convert.join=true; 这样设置，hive就会自动的识别比较小的表，继而用mapJoin来实现两个表的联合。看看下面的两个表格的连接。

```
<property>
    <name>hive.auto.convert.join.noconditionaltask.size</name>
    <value>10000000</value> The default is 10MB
</property>
```

DistributedCache是分布式缓存的一种实现，它在整个MapReduce框架中起着相当重要的作用，他可以支撑我们写一些相当复杂高效的分布式程 这里的第一句话就是运行本地的map join任务，继而转存文件到XXX.hashtable下面，在给这个文件里面上传一个文件进行map join，之后才运行了MR代码去运行计数任务。说白了，在本质上mapjoin根本就没有运行MR进程，仅仅是在内存就进行了两个表的联合。

mapjoin使用场景 1.关联操作中有一张表非常小 2.不等值的链接操作 自动执行

```
set hive.auto.convert.join=true;
hive.mapjoin.smalltable.filesize=25;默认值是25mb
```

```
<property>
  <name>hive.mapjoin.smalltable.filesize</name>
  <value>25000000</value>
 </property>
```

手动执行 A为小表 如果A表超过25M，还想使用map join;

select /+mapjoin(A)/ f.a,f.b from A t join B f on(f.a==t.a) hive入门学习：

join的三种优化方式 Reduce-side Join

hve join操作默认使用的就是reduce join Reduce-side Join原理上要简单得多，它也不能保证相同key但分散在不同dataset中的数据能够进入同一个Mapper，整个数据集合的排序 在Mapper之后的shuffle过程中完成。相对于Map-side Join，它不需要每个Mapper都去读取所有的dataset，这是好处，但也有坏处， 即这样一来Mapper之后需要排序的数据集合会非常大，因此shuffle阶段的效率要低于Map-side Join。

reduce side join是一种最简单的join方式，其主要思想如下： 在map阶段，map函数同时读取两个文件File1和File2，为了区分两种来源的key/value数据对，对每条数据打一个标签（tag） semi join 小表对大表 是reudce join的变种 map阶段过滤掉不需要join的字段 相当于Hivw SQL加的where过滤 SMB Join（sort merge bucket） SMB 存在的目的主要是为了解决大表与大表间的 Join 问题，分桶其实就是把大表化成了“小表”，然后 Map-Side Join 解决之，这是典型的分而治之的思想。

```
1 set hive.enforce.bucketing=true;
2 set hive.enforce.sorting=true;
```

表优化数据目标：相同数据尽量聚集在一起