一、hive概述

1.hive的背景

facebook为结果海量数据分析，避免使用传统mr二开发出来的类sql的操作大数据工具。

2.hive的定义

hive是一个数据仓库软件，它能够使用类sql进行读写，管理基于集群上的海量数据，hive可以对已经存在的数据进行结构化，同时hive也提供命令行和jdbc让用户进行连接。

3.hive和hadoop的关系

hive基于hadoop，hive本身没有存储数据的能力，也没有数据处理能力，仅是将sql转化成执行计划。

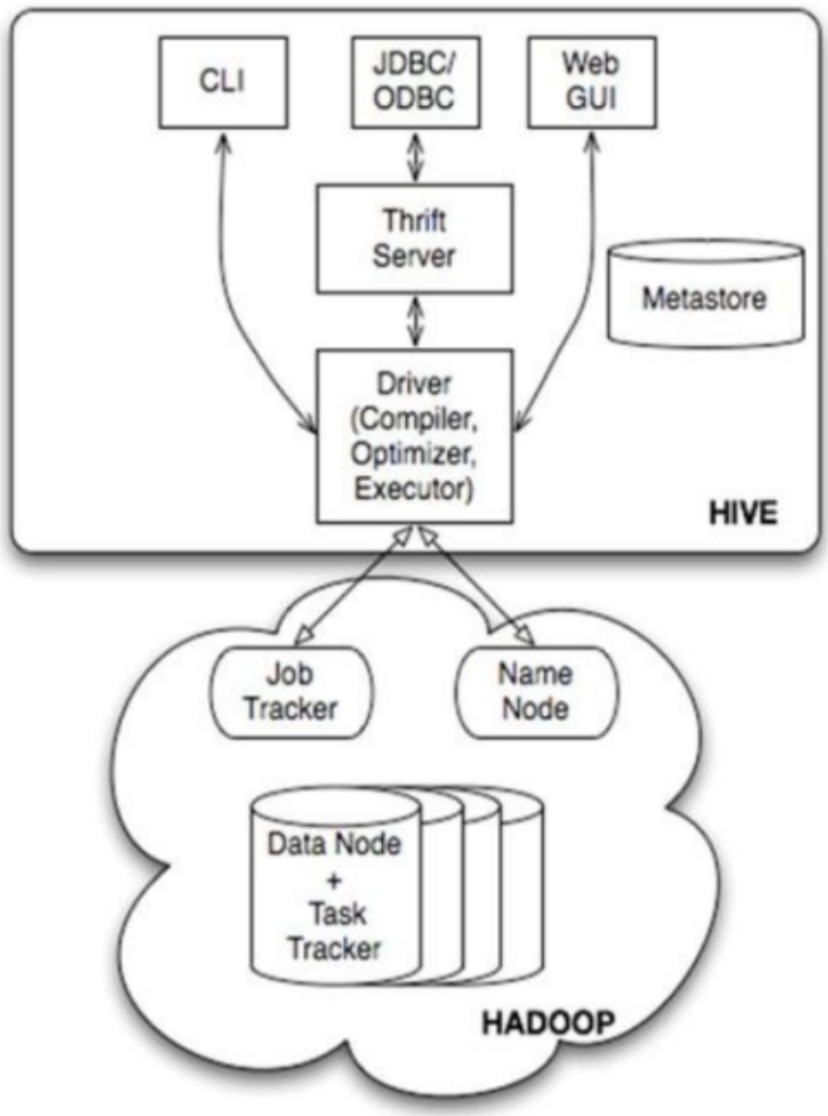
4.hive的特点

```
1.用过sql容易的处理数据
2.对不同的数据进行结构化
3.直接处理存在的hdfs的文件或者其他（如hbase）的数据
4.hive通过spache tez，apache spark，mr进程查询
5.通过HPL-sql去做存储过程
6.通过hive LLAP,Apache YARN和Apache Slider 进行sub-second查询检索。
```

```
延展性
    hive支持用户自定义函数，让用户可以根据自己的需求来是实现自己的函数。
容错性
    节点出现问题sql仍可执行
可扩展
```

5.hive的架构

Hive架构简述



主要分为以下几个部分：

用户接口，包括 命令行CLI，Client，Web界面WUI，JDBC/ODBC接口等

中间件：包括thrift接口和JDBC/ODBC的服务端，用于整合Hive和其他程序。

元数据metadata存储，通常是存储在关系数据库如 mysql，derby 中的系统参数

底层驱动：包括HiveQL解释器、编译器、优化器、执行器（引擎）。

Hadoop：用 HDFS 进行存储，利用 MapReduce 进行计算。

1.用户接口主要有三个：CLI，Client 和 WUI。其中最常用的是 CLI，Cli 启动的时候，会同时启动一个 Hive 副本。Client 是 Hive 的客户端，用户连接至 Hive Server。在启动 Client 模式的时候，需要指出 Hive Server 所在节点，并且在该节点启动 Hive Server。 WUI 是通过浏览器访问 Hive。

2.Hive 将元数据存储在数据库中，如 mysql、derby。Hive 中的元数据包括表的名字，表的列和分区及其属性，表的属性（是否为外部表等），表的数据所在目录等。

3.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、编译生成执行计划、优化以及生成最佳执行计划。生成的查询计划存储在 HDFS 中，并在随后有 MapReduce 调用执行。

4.Hive 的数据存储在 HDFS 中，大部分的查询由 MapReduce 完成（包含 * 的查询，比如 select * from tbl 不会生成 MapRedcue 任务）。

6.hive的元数据存储形式

6.1.derby

derby也是个数据库，是hive内嵌的数据库，存储量偏小，只支持单session

如果不配置用mysql存储元数据的话，当输入hive时候，会在当前目录下创建一个元数据目录，当前目录下不能再次进入hive，但是在其他目录下可以进入。

6.2.mysql

用mysql存储元数据（几个关键库的存储信息要知道）

二、hive的安装部署

1.下载hive的安装包

链接:https://pan.baidu.com/s/16Bo08RnawlS-wNymzxpJ6w 提取码:8u6y

2.解压安装包到某个目录

```
tar -zxvf xxxxx
```

3.配置hive的环境变量

```
vi /etc/profile
export HIVE_HOME=
PATH=PATH:HIVE_HOME/bin
```

4.到hive目录下的conf中，配置hive-site.xml

```
<configuration>
    <!--指定hive的默认数据文件存储格式-->
    <property>
        <name>hive.default.fileformat</name>
        <value>TextFile</value>
    </property>
    <!--元数据连接入口，默认就是9083端口-->
    <property>
         <name>hive.metastore.uris</name>
        <value>thrift://mini1:9083</value>
    </property>
    <!--指定元数据存储的数据库信息-->
    <property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:mysql://mini1:3306/onhive</value>
    </property>
    <!--指定元数据存储的链接驱动-->
    <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>root</value>
    </property>
    <!--配置元数据的远程beeline连接-->
    <property>
        <name>hive.server2.thrift.bind.host</name>
            <value>mini1</value>
    </property>
    <property>
        <name>hive.server2.thrift.port</name>
        <value>10000</value>
    </property>
</configuration>
```

5.把mysql的驱动包放入hive的lib目录下

6.给mysql用户授权

```
grant all privileges on *.* to root@'%' identified by 'root';
flush privileges;
```

7.输入hive测试

注意可能出现jar包版本错误

```
注意:
保证hadoop集群中的jline jar包和所有hive节点的 jline jar包版本保持一致(高版本替换低版本)
hadoop的jline jar包存放位置: HADOOP_HOME/share/hadoop/yarn/lib/下
其中的jline包为: jline-0.9.94.jar(hadoop-2.6.5)
hive的jline jar包存放位置: HIVE_HOME/lib/下
其中的jline包为: jline-2.12.jar(hive-1.2.1)
删除hadoop的jline包, 把hive的jline包放进去
```

要想让集群的其他机器共享该服务器的元数据信息，只需要在其他机器的hive-site.xml中加入如下配置

```
    <!--mini1是主服务器的hive元数据入口地址-->
    <property>
         <name>hive.metastore.uris</name>
        <value>thrift://mini1:9083</value>
    </property>
```

然后再mini1服务器上启动元数据服务

```
hive --service metastore &
```

在其他机器上输入hive测试表信息是否一样。

★hiveserver2的启动方法

```
nohup hiveserver2>s1.log 2>&1 &
beeline
!connect jdbc:hive2://mini1:10000
```

三、hive的语法

1.基本语法

1.1hive的命名规则

```
1.不能使用关键字，数字开始的字符串来做库表名，不区分大小写
2.字段可以使用关键字，但是要加反引号``
```

1.2建表语法

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
   [(col_name data_type [COMMENT col_comment], ...)]
   [COMMENT table_comment]
   [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
   [CLUSTERED BY (col_name, col_name, ...)
   [SORTED BY (col_name [ASC|DESC], ...)] INTO num_buckets BUCKETS]
   [ROW FORMAT row_format]
   [STORED AS file_format]
   [LOCATION hdfs_path]

•CREATE TABLE 创建一个指定名字的表。如果相同名字的表已经存在，则抛出异常；用户可以用 IF NOT EXIST
选项来忽略这个异常
•EXTERNAL 关键字可以让用户创建一个外部表，在建表的同时指定一个指向实际数据的路径（LOCATION）
•LIKE 允许用户复制现有的表结构，但是不复制数据
•COMMENT可以为表与字段增加描述
•PARTITIONED BY 指定分区
•ROW FORMAT
   DELIMITED [FIELDS TERMINATED BY char] [COLLECTION ITEMS TERMINATED BY char]
      MAP KEYS TERMINATED BY char] [LINES TERMINATED BY char]
      | SERDE serde_name [WITH SERDEPROPERTIES
      (property_name=property_value, property_name=property_value, ...)]
   用户在建表的时候可以自定义 SerDe 或者使用自带的 SerDe。如果没有指定 ROW FORMAT 或者 ROW
FORMAT DELIMITED，将会使用自带的 SerDe。在建表的时候，
用户还需要为表指定列，用户在指定表的列的同时也会指定自定义的 SerDe，Hive 通过 SerDe 确定表的具体的列
的数据。
•STORED AS
   SEQUENCEFILE //序列化文件
   | TEXTFILE //普通的文本文件格式
```

```
    | RCFILE    //行列存储相结合的文件
    | INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname //自定义文件格式
    如果文件数据是纯文本，可以使用 STORED AS TEXTFILE。如果数据需要压缩，使用 STORED AS SEQUENCE。
•LOCATION指定表在HDFS的存储路径
最佳实践：
    如果一份数据已经存储在HDFS上，并且要被多个用户或者客户端使用，最好创建外部表
    反之，最好创建内部表。
    如果不指定，就按照默认的规则存储在默认的仓库路径中
```

1.3表数据的修改

```
1.修改表名
    alter table test rename to test1; //可以去数据库的TBLS修改表名
2.修改字段信息
    --把string类型的id修改为int的myid并把它放在name的后面。（数据内容不会修改）
    alter table test change column id myid int after name;
    --把id改为第一列
    alter table test change column myid id string frist;
3.添加字段
    alter table test add column (dt int,dt2 string)
4.删除字段(没有drop功能，只能通过replace来实现)
    --把dt和dt2去除掉
    alter table test replace columns(
        id string,
        name string
    )
5.内外部表的修改
    alter table test set tblproperties('EXTERNAL'='TRUE')--必须大写
    alter table test set tblproperties('EXTERNAL'='false')--可以小写
6.显示当前库
    set hive.cli.print.current.db=true;
7.删除数据库
    drop database test;--里面有表的的话不会删除
    drop database test cascade;--强制删除
```

1.4内外部表的区别

```
1.
    内部表删除的时候，会把元数据和hdfs的目录都删除掉
    外部表删除的时候，只会删除数据库中的元数据信息，hdfs的目录不会被删除，如果再次创建表之后，数据会自动加载（在相同的数据库中，相同的表名）。
2.
    默认创建的是内部表，创建外部表的话需要加上external关键字
3.
    一般使用外部表（长期存在的表，数据量大的，不希望把数据块删除）
    临时表或者确定使用之后即可清空全部数据的话，则可以使用内部表。
```

1.5查看表的信息描述

```
desc test;
show create table test;
desc extended test;
describe table test;
```

###

2.加载数据6种方式

2.1使用load加载

```
//从本地文件加载数据(是文件的复制粘贴)
load data local inpath '/home/hadoop/1.txt' into table test;
//从hdfs加载数据（文件的剪切粘贴)
load data inpath '/hivedata/' into table test;
```

2.2创建表的使用location指定

```
create table if not exists test(
id int comment ' this is test' --注意字段没有默认值default
name string
)
row format delimited fields terminated by ' ' --行字段分割符，不允许使用多个
lines terminated by '\n' --换行符，只有\n
stroed as textfile --文件存储格式，默认是textfile
location '/user/hive/warehouse/test.db/test' --此处只能跟hdsf的目录
;
```

2.3.insert into

```
方法一:
set hive.exec.mode.local.auto=true;--设置本地模式
insert into test
select id,name,sex from test1;

方法二:
from test
insert into table test1
insert overwrite rable test2
select
id,
name
;

(带有动态分区的)
from test
insert overwrite table test1 partition(age)
select name,address,school,age
insert into table test2
```

```
select name,address

方法三:
with tmp as(
select
id,
name
from test
)
insert into table test1
select * from tmp;
```

2.4直接将文件put到表所对应的hdfs目录

```
hdfs dfs -put ./1.txt /user/hive/warehouse/test.db/test
```

2.5.create as

```
create table test1
as
select name from test
;
```

2.6使用like

```
create table test1 like test;--只是创建表并没有把数据加载
create table test1 like test location '/xxx/xxx/' --有数据
```

四、hive的分区表

1.分区表的意义

分区表的意义?

避免全盘扫描，提高查询的效率。

一般使用什么分区?

一般使用日期，地域，能将数据分散开的。

2.分区的本质

在表的目录或是分区的目录下再创建目录，分区的目录名为指定字段=值（比如year=2019）

3.分区注意点

```
1.分区名字区分大小写
2.分区字段是一个伪字段，但是可以用来进行操作
3.一张表可以有一个或者是多个分区，并且分区下面可以有一个多个分区
4.分区字段是表外字段
```

4.创建分区表

4.1一级分区表

```
--建表
create table if not exists test(
id int,
name string
)
partitioned by (time string)
row format delimited fields terminated by ' '
;

--加载数据
--此时是静态分区的加载，后面会说动态分区和混合分区
load data local inpath '/home/hadoop/1.txt' (overwrite)into table test
partition(time='2019')
```

4.2二级分区表

```
--建表
create table if not exists test(
id int,
name string
)
partitioned by (time string,age int)
row format delimited fields terminated by ' '
;

--加载数据
load data local inpath '/home/hadoop/1.txt' overwrite into table test partition
(time='2019',age=20);
```

5.修改分区表的信息

```
1.查看分区
    show partitions test;
2.添加分区
    --添加一个分区
    alter table test add partition(time='2018') lcoaltion '/hdfs文件路径'
    --一次添加多个分区,用空格放入分隔
alter table test add partition(time='2017') partition(time='2016');
3.修改分区名称
    alter table test partition(time='2016') rename to partition(time='2020');
```

```
4.修改分区路径
    alter table test partition(time='2020') set location '/hdfs文件的路径'
    注意几点:
    --修改分区路径实际是把数据库中的sds文件存储位置改了指向地点。
    --修改后hdfs中的time=2020的文件没有 表 指向它。
    --当删除2020表的时候，会删除新指向的文件，旧文件'time=2020'并不会删除
5.删除分区
    --删除一个
    alter table test drop partition(time='2020');
    --删除多个，用逗号分隔
    alter table test drop partition(time='2020'),partition(time='2019');
```

6.分区表类型

6.1三种分区类型

```
静态分区：加载数据到指定分区的值。可以使用load方式加载
动态分区：数据未知，根据分区的值来确定需要创建的分区。不能使用load方式加载
混合分区：静态和动态都有。
```

6.2动态分区

6.2.1动态分区设置

```
hive.exec.dynamic.partition=true 是否允许动态分区;
hive.exec.dynamic.partition.mode=strict/nostrict
    严格模式下，分区时必须至少指定一个静态分区
    可以所有的都为动态分区，但是尽量评估动态分区数量
hive.exec.max.dynamic.partitions=1000最大动态分区数量
hive.exec.max.dynamic.partitions.pernode=100 单个节点允许最大分区数量
```

6.2.2动态分区案例

```
--建表
create table test(
id int,
name string
)
partitioned by (dt string)
row format delimited fields terminated by ' '
;

--加载数据(静态的方法)
load data local inpath '/home/hadoop/1.txt' into table test partition(dt='2019')

--动态的方法加载数据
--select查询中必须包含分区字段
set hive.exec.mode.local.auto=true;
insert into table test partition(dt)
select
```

```
id,
name,
dt
from test1
;
```

6.3混合分区案例

```
create table if not exists dy_part2(
id int,
name string
)
partitioned by (year int,month int)
row format delimited fields terminated by ' '
;

set hive.exec.mode.local.auto=true;
set hive.exec.dynamic.partition.mode=strict;
insert into table dy_part2 partition(year=2019,month)
select
id,
name,
month
from part2
where year=2019
;
```

7.严格模式禁止的五类查询

1.笛卡尔积

```
select
*
from dept d1
join dept d2
;
```

2.不带分区字段过滤的

```
--错误的情况
select
*
from test;

--正确
select
*
from test
where time='2019';
```

3.order不带limit

```
--错误
select
*
from test
order by id desc;
```

4.bigint和string不允许比较

5.bigint和double不允许比较

8.读时模式

```
hive是一个严格的读时模式。写数据不管数据正确性，读的时候，如果不对的用null来代替。
mysql是一种写时模式。写的时候就要检查语法是否正确。
```

五、hive的查询语句

1.sql的语句的执行顺序

```
FROM                        ---map阶段
<left_table>
JOIN                        ---有map端join，也有reduce端join
<join_condition>
<join_type>
 ON
<right_table>
WHERE                       ---map阶段
<where_condition>
GROUP BY                     ---reduce
<group_by_list>
HAVING                         --- reduce
<having_condition>
SELECT                         ---map端\reduce端
DISTINCT
<select_list>
ORDER BY                       ---reduce端
<order_by_condition>
LIMIT                           ---map或者reduce
<limit_number>
```

```
注意点：
1.尽量不要使用子查询，不要用in和not in ，这是全表查询。
2.子查询一般会产生一个job，（只写个select * from 则不会）
3.永远小表驱动大表。（小表放在左边，紧跟from）
```

2.join连接

```
内连接（inner join）
外连接（outer join）
左连接（left join）
左半连接(left semi join)
右连接（right join）
全连接（full join）
---------------------------------------
一般用
left join
inner join == join == 表1，表2
---------------------------------------

----------------------------------------
**
join ： 不加任何的on 或者是where过滤条件时，称之为笛卡尔积。
**
```

```
left semi join 和 left join 区别：

1.都是左表连接，但是semi join 只关联左右两表都有的数据
2.semi join 只是输出左表的信息
3.semi join是left join的一种优化
4.semi join一般用于查询不存在的情况
```

```
inner join 和outer join的区别：

1.分区字段对outer join 中的on条件是无效，对inner join 中的on条件有效
2.有inner join 但是没有full inner join
3.有full outer join但是没有outer join
4.所有join连接，只支持等值连接(= 和 and )。不支持 != 、 < 、> 、 <> 、>=、 <= 、or
5.外连接匹配不到的会用null来代替，而内连接只会查询匹配到的数据
```

3.子查询问题

```
hive对子查询支持不是很友好，特别是 "="问题较多
select
e.*
from emp e
where e.deptno = (
select
d.deptno
from dept d
limit 1
)
;
//以上语句会出错
-------------------------------------------------
```

```
select
e.*
from emp e
where e.deptno in (
select
d.deptno
from dept d
limit 1
)
;
```

4.map端join

```
hive1.x版本已经默认开启。

map-side join:
如果所有的表中有小表，将会把小表缓存内存中，然后在map端进行连接关系查找。
hive在map端查找时将减小查询量，从内存中读取缓存小表数据，效率较快，还省去大量数据传输和shuffle耗时。

//是否开启map端join
set hive.auto.convert.join=true
//到底小表多大才会被转换为map-side join:
set hive.mapjoin.smalltable.filesize=25000000   约23.8MB

以前的老版本，需要添加(/+MAPJOIN(小表名)/)来标识该join为map端的join。hive 0.7以后hive已经废弃，但是仍然管用:

select
/*+MAPJOIN(d)*/  --标识d为小表
e.*
from u1 d
join u2 e
on d.id = e.id
;
```

5.各种by

```
1.group by
        分组。
        一般来说，有group by出现，就会有reduce task
2.distribute by
        分到多个reduce任务
3.order by
        全局排序，保证所有reduce输出结果有序。
4.sort by
        局部排序，只能保证单个redece结果是有序的。
5.cluster by
        兼有distribute by以及sort by的功能，当两者使用的字段相同时 等价于 cluster by。
        cluster by 排序只能只能是升序。
```

```
order by的缺点:

1.由于是全局排序，所以所有的数据会通过一个Reducer 进行处理，当数据结果较大的时候，一个Reducer 进行处理十分影响性能。
2.只要使用order by ，reducer的个数将是1个。
3.当开启MR 严格模式的时候ORDER BY 必须要设置 LIMIT 子句 ，否则会报错
```

6.limit相关配置

```
将set hive.limit.optimize.enable=true 时，limit限制数据时就不会全盘扫描，而是根据限制的数量进行抽样
同时还有两个配置项需要注意:
hive.limit.row.max.size         这个是控制最大的抽样数量 //默认100000
hive.limit.optimize.limit.file 这个是抽样的最大文件数量 //默认10
```

7.union相关说明

```
1.单个union 语句不支持: orderBy、clusterBy、distributeBy、sortBy、limit
2.单个union语句字段的个数要求相同，字段的顺序要求相同、字段类型需要一样(union)。
3.union子句不支持group by\distribute by\order by (每个union子句嵌套中可以使用)
```

六、hive的数据类型

1.hive的基础数据类型

```
tinyint         1          -128-127
smallint        2
int     4
bigint  8
String    可变
boolean  1
double  8
float   4
timestamp       时间格式2019-8-8 12:11:00
binary  字节数组

java中有的数据类型在hive中没有:
long、byte、char、short
```

2.hive的复杂数据类型

2.1Array

案例说明:

```
create table if not exists arr1(
name string,
score Array<int>
)
row format delimited fields terminated by '\t'
collection items terminated by ',' --数组中数据的分割符
;

//加入数据
zhangsan 90,80,99
lisi     80,99,85

//查询
select name,score[0],score[2] from arr1 where score[1]>80;
--结果:zhangsan 80 99

//数组越界不报错
score[3]=null
```

```
//函数统计每个学生的总成绩

select name,sum(cj) from arr1 lateral view explode(score) score as cj group by name;
```

--explode

```
select explode(score) score from arr1;
//结果
90
80
99
80
99
85
```

--lateral view

```
虚拟表
create table temp
as
select name,cj from arr1 lateral view explode(score) score as cj ;


zhangsan        90
zhangsan        80
zhangsan        99
lisi    80
lisi    99
lisi    85
```

--collect_set和collect_list

```
作用:
        聚合成一个集合。
        set会去重,list不会去重

//把上面的temp表，转回arr1表的形式，插入到arr2

create table if not exists arr2(
name string,
score array<int>
)
row format delimited fields terminated by ' '
collection items terminated by ','
;

insert into table arr2
select
name,
collect_list(cj)
from temp
group by name;
```

2.2.map

```
//数据
zhangsan chinese:90,math:87,english:63,nature:76
lisi chinese:60,math:30,english:78,nature:
wangwu chinese:89,math:,english:81,nature:9

//建表
create table if not exists map2(
name string,
score map<string,int>
)
row format delimited fields terminated by ' '
collection items terminated by ','
map keys terminated by ':'
;

//加载数据
load data local inpath '/home/hivedata/map' into table map2;

//查询数学大于35分的学生的英语和自然成绩:
select
name,
score['english'] ,
score['nature']
from map2
where score['math'] > 35
;
```

--展开map

```
select name,a,b from map2 latetal view explode(score) score as a,b ;

结果:
zhangsan        chinese 90
zhangsan        math    87
zhangsan        english 63
zhangsan        nature  76
lisi    chinese 60
lisi    math    30
lisi    english 78
lisi    nature  NULL
wangwu  chinese 89
wangwu  math    NULL
wangwu  english 81
wangwu  nature  9
```

--str_to_map

```
--参数说明
        1.要进行拼接的字符串
        2.每个k-v之间的分割符
        3.kv内部的分隔符
如: select str_to_map("k1:123&k2:234","&",":")

--把上面数据拼回map2表的形式
select
name,
str_to_map(concat_ws('&',collect_list(concat_ws(':',a,b))),'&',':')
from
temp
group by name;
```

2.3struct

```
//创建表
create table if not exists struct1(
name string,
score struct<chinese:double,math:double,english:double>
)
row format delimited fields terminated by '\t'
collection items terminated by ','
;

//加入数据
zhangsan 90,80,99
lisi     80,99,85

//查询
select
name,
score.chinese,
```

```
score.math
from struct1
where score.math>90
;
--结果: lisi 80.0 85.0
```

结构体和array的区别:

1.数组只能是某种类型的元素集合，但是结构体可以任意类型
2.array的函数较为丰富，结构体相对较少
3.两者取值不一样，array使用角标取值，而struct使用 .属性名取值。

七、hive的分桶

1.分桶的意义

1.当单个的分区或者表的数据量过大，分区不能更细力度的划分数据，就需要使用分桶技术划分成更细的力度。

2.为了保存分桶查询的分桶结构（数据按照分桶字段进行保存hash散列） 3.分桶表的数据进行抽样和JOIN时可以提高查询效率 sample

2.分桶案例

默认，分桶技术实现是按照分桶字段进行hash值，然后%总的分桶数，即是分桶号。

```
create table if not exists buc1(
id int,
name string,
age int
)
clustered by (id) into 4 buckets
row format delimited fields terminated by ','
;


数据:
id,name,age
1,aa1,18
2,aa2,19
3,aa3,20
4,aa4,21
5,aa5,22
6,aa6,23
7,aa7,24
8,aa8,25
9,aa9,26

//分桶表不允许用load方式粗暴加载，加载会成功，但是不会建立分桶文件。
load data local inpath '/home/hivedata/buc1.txt' into table buc1;
----------------------------------------------------------------------
```

```
//分桶表使用insert into的方式加载
---------------------------创建临时表---------------------
create table if not exists buc2(
id int,
name string,
age int
)
row format delimited fields terminated by ','
;
--------------------加载数据到临时表-----------------------------
load data local inpath '/home/hivedata/buc1.txt' into table buc1;
--------------------- 使用分桶查询将数据导入到分桶表---------------
insert overwrite table buc2
select id,name,age from buc1
cluster by (id)    --等价 distribute by (id) sort by (id asc)
;
--------------设置强制分桶的属性-------------------
set hive.enforce.bucketing=false/true
<name>hive.enforce.bucketing</name>
<value>false</value>
<description>Whether bucketing is enforced. If true, while inserting into the table,
bucketing is enforced.</description>

------------如果设置了reduces的个数和总桶数不一样，请手动设置:------------
set mapreduce.job.reduces=?
--------------------------------------------------------------------------

tablesample (bucket x out of y on id)
1.x不能大于y
2.x从哪一个桶开始取数，y代表总的桶数（一般是桶的倍数或者因子）

总：分桶字段%y+1 = x 的值 j将会被查询出来。

------------------------查看结果----------------------------
 //全查询，相当于 select * from buc2;
 select * from buc2 tablesample(bucker 1 out of 1 on id);

 4
 8
 1
 5
 7
 2
 6
 3
 7

 //某个桶
        select * from buc2 tablesample(bucket 1 out of 4 on id);
 //1和3桶
        select * from buc2 tablesample(bucket 1 out of 2 on id);
```

```
    select * from buc2 tablesample(bucket 1 out of 8 on id);

 select
 *
 from buc2
 tablesample(bucket 2 out of 3 on id)
 where id > 5
 ;
 查出结果:     7
```

```
//随机查询3条
select * from buc1 order by rand() limit 3;
```

```
//只查前三行，不随机（没啥意义）
select * from buc1 table tablesample(3 rows);
```

```
//查询百分之前多少
select * from buc1 table tablesample(30 percent);
```

```
//取3k   单位--B\K\M\G
select * from buc1 table tablesample(3k);
```

八、hive的函数

1.hive的内置函数

查询函数语句：

```
显示所有函数:
show functions;
查看函数描述:
desc functionfunc_name;
模糊查找hive的函数:
show functions like '*concat*';
```

```
rand(7)
round(double,n)   //四舍五入
split(str,spliter)
substr(str,start,len) 下标从1开始
concat(str,str1...)
concat_ws("",str,str1) //输入类型是字符串  或者 一个数组
*********************************************************************************
cast(col as type)
```

```
**********************************************************************************************
is null \ is not null
**********************************************************************************************
nvl(1,2)如果表达式1为空值,NVL返回值为表达式2的值,否则返回表达式1的值
**********************************************************************************************
if(condition,true,false)
coalesce(v1,v2,v3) //返回第一个不为空的
**********************************************************************************************
case col
when value then ...
when value then ...
else ...
end

case
when col=value then...
when col=value1 then...
else ...
end
**********************************************************************************************
*****
size(arr/map)
length(str)
**********************************************************************************************
*****
select "1"+"abc" // null
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!关于日期的函
数!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
to_date(): 返回 yyyy-MM-dd日期格式  select to_date('2015-04-02 13:34:12'); //2015-04-02
**********************************************************************************************
*****
from_unixtime('时间戳','yyy-MM-dd hh:mm:ss')   //select
from_unixtime(1323308943,'yyyyMMdd'); //20111208
unix_timestamp('2019-09-08','格式')    //查询时间戳， 可以不写参数


select unix_timestamp();
输出: 1430816254
select unix_timestamp('2015-04-30 13:51:20');
输出: 1430373080
//两个时间小时差
select (unix_timestamp('2018-05-25 12:03:55') - unix_timestamp('2018-05-25
11:03:55'))/3600 //1.0

**********************************************************************************************
*****
思想：先转换成时间戳，再由时间戳转换为对应格式。
--20171205转成2017-12-05
select from_unixtime(unix_timestamp('20171205','yyyymmdd'),'yyyy-mm-dd') from dual;
--2017-12-05转成20171205
select from_unixtime(unix_timestamp('2017-12-05','yyyy-mm-dd'),'yyyymmdd') from dual;
**********************************************************************************************
*****
```

```
weekofyear() // 日期在当前年的周数   //select weekofyear('2015-05-05 12:11:1'); //输出: 19
*********************************************************************************************
*****
date_diff: 返回开始日期减去结束日期的天数  //select datediff('2015-04-09','2015-04-01');  //8
----应用----
计算某一个日期属于星期几，如2018-05-20 是星期日
SELECT IF(pmod(datediff('2018-05-20', '1920-01-01') - 3, 7)='0', 7, pmod(datediff('2018-
05-20', '1920-01-01') - 3, 7))
输出: 7
------------

date_sub:  返回日期前n天的日期     //select date_sub('2015-04-09',4); //输出: 2015-04-05
date_add:返回日期后n天的日期       //select date_add('2015-04-09',4); //输出: 2015-04-13
last_day: 获取月末最后一天        // select last_day('2018-09-20');  //2018-09-30
trunc()            //select trunc('2018-09-27','YY') 获取年初，月初同方法（MM）。
next_day(,)       //可以求出当前日期的下个星期一。select next_day( sysdate, 'MONDAY') from
dual;

current_date(): //当前日期
current_timestamp()://select current_timestamp();//2019-09-12 04:37:39.866
hour()           //日期的小时

base64():
加解密场景:
保护数据、将数据变得更小


encode():编解码
decode():

format_number(,保留位数):格式化数字


instr():select instr('aaasss','s'); //;
trim():
ltrim():
lpad():左补足 //select lpad('aaasss',9,'q'); qqqaaasss

replace(str,'',''):
substring_index(str,'a',2):指定索引位置
initcap():首字符变大写
find_in_set():查找是否在set中，费性能
str_to_map():转map


-----------------------------------------------------
initcap():首字符变大写
select initcap('yuniko');
-----------------------------------------------------
find_in_set():查找是否在set中，费性能
eg: select find_in_set('1','2,1,2,3');
eg: select find_in_set('1','2,2,2,3'); //没有输出0
-----------------------------------------------------
```

```
get_json_object(json,'$.xxx'):获取json对象
**********************************************************************************
regexp_extract(string subject,  string pattern,  int index):正则抽取  ()()()
//返回值: string
//说明:   将字符串subject按照pattern正则表达式的规则拆分，返回index指定的字符。
regexp_extract('x=a3&x=18abc&x=2&y=3&x=4','x=([0-9]+)([a-z]+)',0),  -- x=18abc
regexp_extract('x=a3&x=18abc&x=2&y=3&x=4','^x=([a-z]+)([0-9]+)',0), -- x=a3

regexp_extract('https://detail.tmall.com/item.htm?
spm=608.7065813.ne.1.Ni3rsN&id=522228774076&tracelog=fromnonactive','id=([0-9]+)',0),
-- id=522228774076
regexp_extract('https://detail.tmall.com/item.htm?
spm=608.7065813.ne.1.Ni3rsN&id=522228774076&tracelog=fromnonactive','id=([0-9]+)',1),
-- 522228774076

regexp_extract('http://a.m.taobao.com/i41915173660.htm','i([0-9]+)',0),            --
i41915173660
regexp_extract('http://a.m.taobao.com/i41915173660.htm','i([0-9]+)',1)             --
41915173660
**********************************************************************************
regexp_replace(str,'',''):正则替换 //regexp_replace(string A, string B, string C) 字符串替换
函数，将字符串A 中的B 用 C 替换
**********************************************************************************
parse_url():解析url

select parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'HOST'); --返回值为
facebook.com
select parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'PATH');-
-/path1/p.php
select parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'QUERY');--
k1=v1&k2=v2
select parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'REF');--Ref1
select parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'PROTOCOL');--http
select parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'AUTHORITY');--
facebook.com
select parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'FILE');-
-/path1/p.php?k1=v1&k2=v2
select parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'USERINFO');--null
**********************************************************************************
size(a1):
length(str):
**********************************************************************************
                                map_keys()
                                map_values()
假设有一张表，表名为t，其中字段params的数据类型是map，其map的具体k-v对如下:
{'k0':'abc','k1':'01,02,03','k2':'456'}
-----------------------------------------
1. size(Map)函数: 可得map的长度。返回值类型: int
select size(map(t.params));
>> 3
-----------------------------------------
```

```
2. map_keys(Map)函数：可得map中所有的key；  返回值类型：array

select map_keys(map(t.params));
>> ["k0","k1","k2"]
-----------------------------------------
3.map_values(Map)函数：可得map中所有的value；返回值类型：array
select map_value(map(t.params));
>> ["abc","01,02,03","456"]
-----------------------------------------
4.判断map中是否包含某个key值：
select array_contains(map_keys(t.params),'k0');
>> true
-----------------------------------------
5. 在k-v对中，若value有多个值的情况，如 {'k1':'01,02,03'} ，如果要用 'k1' 中 '02'作为过滤条件，则语句如下：
（这里用到split来处理）
select *
from t
where split(t.params['k1'],',')[1]
>> 02
-----------------------------------------
 6.如果过滤条件为：k2的值必须为'45'开头，则语句如下：
（这里用到substr方法来处理，这里注明一下，1和2分别表示起始位置和长度）
select *
from t
where substr(t.params['k2'],1,2) = '45'
*****************************************************************************************


collect_set():转换成集合，去重
collect_list():转换成集合，不去重

cast():

bin():转换为二进制

==============================
```

2.窗口函数

1.关键字：over

```
over(partition by a,b)
count(a) over(partition by a,b order by a desc)
over(order by a)
```

2.窗口：物理窗口和逻辑窗口 物理窗口：rows 逻辑窗口：range

3.语法： ROWS/range between [CURRENT ROW | UNBOUNDED PRECEDING | [num] PRECEDING] AND [UNBOUNDED FOLLOWING | [num] FOLLOWING| CURRENT ROW]

2 preceding ======向前两行

2 following =======向后两行

current row=======当前行

unbounded preceding ======无上限，一直到最前面

unbounded following =======无下限，一直到最后

4.案例

```
案例:
create table if not exists win(
name string,
class string,
score int
)
row format delimited fields terminated by ' '
;


load data local inpath '/home/hivedata/win' into table win;

逻辑:
select
name,
class,
score,
sum(score) over(order by score range between 5 preceding and 5 following) mm
from win;

物理:
select
name,
class,
score,
sum(score) over(order by score rows between 2 preceding and 3 following) mm
from win;

每个班前一名和其后一名的分差:
select
name,
class,
score,
(score-lag(score,1) over(partition by class order by score asc)) diff
from win;



第一个和最后一个值:
select
name,
class,
score,
first_value(score) over(partition by class order by score) first,
```

```
last_value(score) over(partition by class order by score) last
from win;
```

3.排名函数

三种排名函数

```
row_number()
rank()
dense_rank()
```

三者区别:

```
row_number() over() : 排名函数，名次不会重复，适合于生成主键或者不并列排名
rank() over() :  排名函数，有并列名次，名次不连续。如:1,1,3
dense_rank() over() : 排名函数，有并列名次，名次连续。如: 1, 1, 2
```

案例

```
根据class分组，并根据score排名:

select
class,
name,
score,
row_number() over(distribute by class sort by score desc) rn,
rank() over(distribute by class sort by score desc) rk,
dense_rank() over(distribute by class sort by score desc) drk
from win
;
```

4.聚合函数

```
group by :
grouping sets() : 指定分组
with cube:  数据魔方，任意维度的组合查询
with rollup :
```

4.1.group by

```
分组的意思，使用GROUP BY时，除聚合函数外其他已选择列必须包含在GROUP BY子句中，否则会报错。

求某公司、某部门的员工数量:
select
class,
score,
count(*)
from win
```

```
group by class,score
;
```

注意:
1、有group by时，查询字段要么在group by子句中，要么在聚合函数中。
2、group by与聚合函数搭配使用，但是聚合函数不能嵌套使用，如sum(count(*))。
3、如果group by的列值中有null，则包含该null的行将在聚合时被忽略。为了避免这种情况，可以使用COALESCE来将null替换为一个默认值。
4、group by与聚合函数count()搭配使用时，同时COUNT又和DISTINCT搭配使用时，Hive将忽略对reducer个数的设置(如：set mapred.reduce.tasks=20;)，仅会有一个reducer！!!此时reduce将成为瓶颈，这时我们可以使用子查询的方式解决该问题，同时子查询需要别名。
5、collect_set() 和collect_list() 不是聚合函数，不需要和group by搭配使用

4.2grouping sets

grouping sets可以实现对同一个数据集的多重group by操作。事实上grouping sets是多个group by进行union alll操作的结合，它仅使用一个stage完成这些操作。grouping sets的子句中如果包换()数据集，则表示整体聚合。多用于指定的组合查询。

例:
1、grouping sets(a,b) ==> group by a union all group by b
2、grouping sets(a,b,(a,b) ==> group by a union all group by b union all group by a,b
3、grouping sets(a,b,(a,b),()) ==>group by a union all group by b union all group by a,b union all 无group by语句

案例

```
select
class,
NULL as score,
count(*)
from win
group by class

union all

select
NULL AS class,
score,
count(*)
from win
group by score

union all

select
class,
score,
count(*)
from win
group by class,score
```

```
;

==如上的语句等价于如下的语句

select
class,
score,
count(*)
from win
group by class,score
grouping sets(class,score,(class,score))
;

上面两个语句的结果都如下：

NULL    45      1
NULL    55      1
NULL    74      1
NULL    78      1
NULL    80      1
NULL    92      1
NULL    95      2
NULL    99      2
1       NULL    3
2       NULL    2
3       NULL    5

1       80      1
1       95      2
2       74      1
2       92      1
3       45      1
3       55      1
3       78      1
3       99      2
```

4.3cube

分组个数：2^（分组字段数）

```
cube俗称是数据立方，它可以时限hive任意维度的组合查询。即使用with cube语句时，可对group by后的维度做任意组合查询，如：group a,b,c with cube ,则它首先group a,b,c 然后依次group by a,c 、 group by b,c、group by a,b 、group a 、group b、group by c、group by () 等这8种组合查询，所以一般cube个数=2^3个。2是定值，3是维度的个数。多用于无级联关系的任意组合查询。

select
class,
score,
count(*)
from win
group by class,score
with cube
;
```

```
NULL    NULL    10
NULL    45      1
NULL    55      1
NULL    74      1
NULL    78      1
NULL    80      1
NULL    92      1
NULL    95      2
NULL    99      2
1       NULL    3
1       80      1
1       95      2
2       NULL    2
2       74      1
2       92      1
3       NULL    5
3       45      1
3       55      1
3       78      1
3       99      2
```

4.4rollup

分组个数：分组字段数+1

卷起的意思，俗称层级聚合，相对于grouping sets能指定多少种聚合，而with rollup则表示从左往右的逐级递减聚合，如:group by a,b,c with rollup 等价于 group by a, b, c grouping sets((a, b, c), (a, b), (a), ( )).直到逐级递减为()为止,多适用于有级联关系的组合查询，如国家、省、市级联组合查询。

```
select
class,
score,
count(*)
from win
group by class,score
with rollup
;

NULL    NULL    10
1       NULL    3
1       80      1
1       95      2
2       NULL    2
2       74      1
2       92      1
3       NULL    5
3       45      1
3       55      1
3       78      1
3       99      2
```

4.5Grouping_ID

对于每一列，如果这列被聚合过则返回0，否则返回1。

```
grouping sets/cube/rollup三者的区别：

注：
grouping sets是指定具体的组合来查询。数据集个数根据指定来
with cube 是group by后列的所有的维度的任意组合查询。数据立方体个数维度的的2次方个
with rollup 是group by后列的从左往右逐级递减的层级组合查询。数据集个数维度+1个
cube/rollup 后不能加()来选择列，hive是要求这样。
```

4.6聚合优化

```
hive.map.agg=true;      #最好将其设置为true，因为会用到map端聚合。
hive.new.job.grouping.set.cardinality=30;    #在grouping sets/cube/rollup中是否启用新的job来执行，主要是因为如果分组多而造成数据立方体炸裂，适当设置该阀值，默认为30.
```

5.自定义函数

5.1常见的自定义函数

```
udf：用户自定义函数，user defined function。一对一的输入输出。（最常用的）。
udaf：用户自定义聚合函数。user defined aggregate function。多对一的输入输出。
udtf：用户自定义表生成函数。user defined table-generate function.一对多的输入输出。

1、继承UDF，重写evaluate()，允许重载。（常用）
2、继承genericUDF，重写initlizer () 、getdisplay () 、evaluate()。
```

5.2自定义函数的部署方法

临时部署

```
//临时部署
1.先把jar包放在服务器上（/home/hadoop/hivejar）
2.进hive，输入add jar /home/hadoop/hivejar/xx.jar
3.create temporary function my_concat as '包名.类名'
4.show functions like 'my_concat' 查看是否存在
5.测试函数select my_concat('a','b');

-----------------------------------------
1、将编写的udf的jar包上传到服务器上
2、创建文件
        vi ./hive-init
        add jar /root/1701Demo-0.0.1.jar;
        create temporary function my_concat as '包名.类名';
```

```
3、启动hive的时候带上初始化文件：
        hive -i ./hive-init
-------------------------------------------
1、将编写的udf的jar包上传到服务器上
2、在hive的安装目录的bin目录下创建一个配置文件，文件名：.hiverc
        vi ./bin/.hiverc
        add jar /root/1701Demo-0.0.1.jar;
        create temporary function my_concat as '包名.类名';
3、启动hive
        hive
-------------------------------------------
把jar包 先放在此目录下，然后再hive中直接使用jar包即可。
        <name>hive.aux.jars.path</name>
        <value>$HIVE_HOME/auxlib</value>
```

永久部署

```
1.把jar包放到hdfs
2.进入hive，输入
create function my_concat as '包名.类名' using jar hdfs://mini1:9000/xxx.jar;

以后每次将会加载方法。
注意，该方式创建的函数名是 ==》 库名.函数名，在别的库用此函数时候要加上库名。
```

九、hive的SerDe（数据格式）

1.csv格式

```
create table if not exists csv1(
uid int,
uname string,
age int
)
row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
stored as textfile
;

--加载数据
load data local inpath '/hivedata/csv1' into table csv1;

--
create table if not exists csv3(
uid int,
uname string,
age int
)
row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
with serdeproperties(
```

```
"separatorChar"=",",
"qutoeChar"="'",
"escapeChar"="\\"
)
stored as textfile
;
```

2.json serde

```
添加第三方jar包:
add jar /xx/json-serde-1.3-jar-with-dependencies.jar;

-------------------------------------------------
create table if not exists json1(
name string,
qq string,
score array<map<string,array<int>>>
)
row format serde 'org.openx.data.jsonserde.JsonSerDe'
stored as textfile
;
-------------------------------------------------
//加载数据  josn文件
load data local inpath '/home/hadoop/json' into table json1;
//json文件内容
{"name":"aaa","qq":"111111","score":[{"lyf":[11,22,33],"lyf2":[22,33,44]} ]}
*********************************************************************************
create table if not exists complex(
uid int,
uname string,
belong array<String>,
tax map<String,array<double>>
)
row format serde "org.openx.data.jsonserde.JsonSerDe"
;

//数据
{"uid":1,"uname":"zs","belong":["zs1","zs2","zs3"],"tax":{"shebao":
[220,280,300],"gongjijin":[600,1200,2400]}}
{"uid":2,"uname":"ls","belong":["ls1","ls2","ls3"],"tax":{"shebao":
[260,300,360],"gongjijin":[800,1600,2600]}}

//加载数据
load data local inpath '/home/hivedata/complex' into table complex;

select
c.*
from complex c
where size(c.belong) = 3
and c.tax["gongjijin"][1] > 1800
;
```

3.refex serde

```
//数据
220.181.108.151 [31/Feb/2018:00:02:32 +6666]
221.181.108.151 [21/Feb/2019:10:02:32 +6666]

//只有满足正则的才能被输出出来，否则为null
create table if not exists regex1(
host string,
dt string
)
row format serde 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES(
"input.regex" = "^([0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}) (.*)$"
)
stored as textfile
;
**********************************************************************************************
***************
hive对文件中字段的分隔符默认情况下只支持单字节分隔符，如果数据中的分隔符是多字节的，则hive默认是处理不了的。
01||gaoyuanyuan||18
02||gaogao||26
create table if not exists bi(
id int,
name string,
age int
)
row format delimited fields terminated by '%'
stored as 'customInputFormat'
;

解决办法，用正则的方式

create table if not exists t_bi_reg(
id string,
name string,
age int
)
row format serde 'org.apache.hadoop.hive.serde2.RegexSerDe'
with serdeproperties(
'input.regex'='(.*)\\|\\|(.*)\\|\\|(.*)',
'output.format.string'='%1$s %2$s %3$s'
)
stored as textfile
;

load data local inpath '/hivedata/bi.txt' into table t_bi_reg;



缺点：字段多的情况，要指定的内容也多
```

通过自定义InputFormat来解决特殊字符分割的问题。
原理是在InputFormat读取行时将数据中的"多字符分隔符"替换为hive的默认的单字符分隔符，以便hive在执行serde操作时可以按照默认的分隔符的格式来对字段进行分割。

可以自定义hive的输入输出

建表：
```
create table if not exists t_bi_reg(
id string,
name string,
age int
)
row format delimited fields terminated by '|'
stored as inputformat 'com.bigdata.hiveUDF.inputformat.BiDelimiterInputFormat'
stored as outputforma 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
;
```

十、hive的文本存储格式

```
1.textfile：默认格式，普通文本文件，数据不压缩，磁盘开销大，分析开销大，查询效率低下。
          少量数据可以使用
          可以配合压缩属性压缩
2.sequencefile：hive提供的一种二进制存储格式，可以切割，天生压缩

          1.不可以直接使用load加载
          2.可以使用insert into table sf select name,score from xx;
          3.占内存比textfile大，压缩算法本身占一定空间
          4.查询速度比textfile格式快。

3.rcfile：hive提供的一种行列混合存储方式，该方式将会把相近的行和列数据放在一块
存储较耗时，查询效率高，也天生压缩。

          1.不能用load加载。
          2.比sequencefile占空间稍微小一些。
          3.查询速度比textfile格式快。

4.orc：是rcfile的一种优化存储。
          1.可以用load加载，但是select * from xx; 读不出来。
          2.查询速度比rcfile快。

5.parquet：面向列存储，可存json格式

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



map端输出压缩：
mapreduce.output.fileoutputformat.compress=false
mapreduce.output.fileoutputformat.compress.codec=org.apache.hadoop.id.compress.DefaultCod
```

```
ec

reduce输出压缩（reduce压缩）：
snappy、bzip2、gzip2、DefaultCompress
mapreduce.output.fileoutputformat.compress=false
mapreduce.output.fileoutputformat.compress.type=NONE/RECORD/BLOCK
mapreduce.output.fileoutputformat.compress.codec=org.apache.hadoop.io.compress.DefaultCod
ec

hive压缩配置:
hive.exec.compress.output
hive.exec.compress.intermediate
hive.intermediate.compression.codec
hive.intermediate.compression.type
```

案例

```
CREATE TABLE `u4`(
  `id` int,
  `name` string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
stored as textfile;

set mapreduce.output.fileoutputformat.compress=true;
set hive.exec.compress.output=true;
insert into table u4
select * from u2;

2、sequence ：
CREATE TABLE `u4`(
  `id` int,
  `name` string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
stored as sequencefile;

3、rcfile ：
CREATE TABLE `u5`(
  `id` int,
  `name` string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
stored as rcfile;

4、orc ：
CREATE TABLE `u6`(
  `id` int,
  `name` string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
stored as orc;

5、parquet
CREATE TABLE `u7`(
  `id` int,
  `name` string)
```

```
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
stored as PARQUET;
insert into table u7
select * from u2;

自定义:
数据:
seq_yd元数据文件:
aGVsbG8gemhhbmdoYW8=
aGVsbG8gZmVpZmVpLGdvb2QgZ29vZCBzdHVkeSxkYXkgZGF5IHVw
seq_yd文件为base64编码后的内容，decode后数据为:

create table cus(str STRING)
stored as
inputformat 'org.apache.hadoop.hive.contrib.fileformat.base64.Base64TextInputFormat'
outputformat 'org.apache.hadoop.hive.contrib.fileformat.base64.Base64TextOutputFormat';

LOAD DATA LOCAL INPATH '/home/hivedata/cus' INTO TABLE cus;
```

十一、hive的视图和索引

1.视图

hive的视图简单理解为逻辑上的表 hive现目前只支持逻辑视图，不支持物化视图。

1.1.hive的视图意义

1、对数据进行局部暴露(涉及隐私数据不暴露)。 2、简化复杂查询。

1.2.视图的简单操作

```
创建视图
create view v1 as
select name,age from table1;

显示视图
show tables; --视图表会出现在里面

显示描述
desc v1;
desc extended v1;
show create table v1;

删除试图
//删除原有表之后，视图就不能使用了。
drop view if exists tab_v2;
drop table if exists tab_v1;   （不支持）
```

1.3.注意点

注意：
1.删除视图对应的表后再去使用视图出错。
2.视图不能用load或者insert into 方式来加载数据。
3.视图只是有可读的属性，不能去修改。

2.索引

索引是数据库标配的技术，hive索引从0.7以后才开始支持的。

2.1.索引的优缺点

优点：避免全表扫描或者减小扫描数据流，提高查询效率
缺点：将会有冗余存储，加载数据耗时

2.2索引特点

索引文件本身有序，索引文件较小

2.3.索引简单操作

```
//查看索引
show index on log1;

//创建索引
create index idx_log1_pho
on table log1(phonenumber)
as 'compact'
with deferred rebuild
;


//重新构建索引
//会跑一个MapReduce
alter index idx_log1_pho
on log1 rebuild;

//创建联合索引
create index idx_log1_union
on table log1(id,phonenumber)
as 'compact'
with deferred rebuild;


//创建bitmap索引
create index idx_log1_upflow
on table log1(upflow)
as 'bitmap'
with deferred rebuild;
```

```
//删除索引
drop index if exists idx_log1_upflow on log1;
```

十二、hive的日志

```
hive的系统日志:
默认目录: /tmp/{user.name}
hive.log.dir={java.io.tmpdir}/{user.name}
hive.log.file=hive.log
hive的查询日志:
<name>hive.querylog.location</name>
<value>{system:java.io.tmpdir}/${system:user.name}</value>
<description>Location of Hive run time structured log file</description>
```

十三、hive的运行方式

```
1、cli :  命令行(hive/beeline)  如果启动beeline连接需要启动hiveserver2
hive --service hiveserver2 &
hiveserver2 &

beeline 可以设置是否启用用户密码，用户权限设置。

2、java的jdbc连接运行
        idea代码。
3、hive -f hql文件

4、hive -e 查询语句
    //在hive外面执行
    hive --database aaa --hivevar min_limit=1 -e 'select * from b limit
${hivevar:min_limit}'
```

-f、-e

```
//把查询出的数据导出到/home/hadoop/aaa
insert overwrite local directory '/home/hadoop/aaa'
row format delimited fields terminated by '\t'
select * from aa
where id > 5
limit ${hivevar:m_lim}

//执行
hive --database xxx --hivevar m_lim=2 -f /home/hadoop/aaa.hql

//也可以用这种发放将数据导出到本地
hive --database xxx --hivevar min_limit=1 -e 'select * from u3 limit
${hivevar:min_limit}'>/home/hadoop/bbb
```

```
注意:
1、一个--hivevar 或者 --hiveconf 只能带一个参数
2、--hiveconf 或者 --hivevar 可以混合使用
3、--hiveconf 或 --hivevar 定义参数不能取消
```

hive的jdbc

```
1、conn、ps\rs的关闭顺序需要时rs\ps\conn,否则报错sasl
2、连接的用户名和密码需要 填写，如果没有配置可以使用root、root,否则会报错没有权限。
```

kylin： 加速hive的查询(加查询预执行，并将结果保存hbase中)

TestJDBC

```
package hive.jdbc;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

/**
 * hive的连接
 */
public class HiveJDBC {
    private static Connection conn = null;
    public static void main(String[] args) {
        selectDemo();
    }

    //
    public static void selectDemo(){
        conn = HiveJdbcUtil.getConn();
        PreparedStatement ps = null;
        ResultSet rs = null;
```

```
        try {
            //获取ps
            ps = conn.prepareStatement("select * from test");
            //为占位符赋值
//            ps.setInt(1,1);
            //执行查询
            rs = ps.executeQuery();
            //遍历结果集
            while (rs.next()){
                System.out.println(rs.getInt(1)+"\t"+rs.getString(2));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            HiveJdbcUtil.closeConn(null,null,rs);
            HiveJdbcUtil.closeConn(null,ps,null);
            HiveJdbcUtil.closeConn(conn,null,null);
        }
    }

}
```

JdbcUtil

```
package hive.jdbc;

import java.sql.*;

/**
 * 连接工具类
 */
public class HiveJdbcUtil {
    private static String driverName = "org.apache.hive.jdbc.HiveDriver";
    private static String url = "jdbc:hive2://mini1:10000/a24"; //指定数据库为dw_hivedata
    private static String userName = "root"; //hiveserver2的用户名和密码
    private static String password = "";
    public static void main(String[] args) {
        System.out.println(getConn());
    }

    /**
     *获取hive的驱动连接
     * @return
     */
    public static Connection getConn(){
        Connection conn = null;
        try {
            Class.forName(driverName); //加载驱动
            //获取conn
            conn = DriverManager.getConnection(url, userName, password);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
```

```
            System.exit(1);
        } catch (SQLException e){
            e.printStackTrace();
        }
        return conn;
    }

    /**
     * 关闭连接
     * @param conn
     */
    public static void closeConn(Connection conn, PreparedStatement ps , ResultSet rs){
        if(conn != null){
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        if(ps != null){
            try {
                ps.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        if(rs != null){
            try {
                rs.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

十四、hive的优化

```
1、考虑环境  （硬件服务器、配置）
2、业务      （统计指标的实现思路）
3、代码或者配置属性
```

1.通过explain或者explain extended来查看执行计划。

```
explain extended
select
id id,
count(id) cnt
```

```
from u4
group by id;
```

以上两种方法都是查看执行计划，只不过extended会打印语句的抽象语义树。
stage:
(1)一个stage相当于一个MapReduce任务或者是mr的一部分（可以是一个子查询，可以是一个抽样，可以是一个合并、可以是一个limit）
(2)hive默认每次只执行一个stage，但是没有依赖关系的可以并行执行。
(3)一个hive的hql语句包含一个或者多个stage，多个之间依赖越复杂，表示任务越复杂，执行效率较低。

2.数据倾斜

数据倾斜：由于key分布不均匀造成的数据向一个方向偏离的现象。
本身数据就倾斜
join语句容易造成
count(distinct col) 很容易造成倾斜
group by 也可能会造成

倾斜现象：
卡在某一个reduce任务的某个进度。

解决方法：
1、找到造成数据倾斜的key，然后再通过hql语句避免(查看日志是哪个task失败--->找该task中关联字段、group by\count(distrinct col) ---> 抽样字段个数 ---> 判断是否是倾斜的key)。单独拿出来处理，然后在和正常的结果进行union all。

2、造成倾斜的key加随机数(加的随机不能造成二次倾斜、保证加随机不能影响原有的业务)。

3、设置相关倾斜的属性
hive.map.aggr=true;
hive.groupby.skewindata=false; （建议开启）
hive.optimize.skewjoin=false;
skewjoin 先关属性查看：
skew 相关的属性：

4、如上都不行，则需要从新查看业务，优化语句流程。

3.join设置

hive的查询永远是小表(结果集)驱动大表(结果集)
hive中的on的条件只能是等值 and连接。on后的比较的两个字段的数据类型尽量相同
注意hive是否配置普通join转换成map端join、以及mapjoin小表文件大小的阀值

注意hive的倾斜join：
hive.optimize.skewjoin=false
hive.skewjoin.key=100000
hive.skewjoin.mapjoin.map.tasks=10000

4.limit的优化

```
//优化是否开启
//如果limit较多时建议开启
hive.limit.optimize.enable=false;

//控制最大的抽样数量
hive.limit.row.max.size=10000;

//抽样的最大文件数量
hive.limit.optimize.limit.file=10;

//fechquery获取最大的行数
hive.limit.optimize.fetch.max=50000;
```

5.本地模式

```
hive查询数据依然还是依靠hadoop。
//是否开启本地模式
hive.exec.mode.local.auto=false;
hive.exec.mode.local.auto.inputbytes.max=134217728;
hive.exec.mode.local.auto.input.files.max=4;
```

6.并行执行

```
hive没有相互依赖的任务可以并行执行。
//是否设置并行执行
hive.exec.parallel=false;

//并行执行线程数
hive.exec.parallel.thread.number=8;
```

7、严格模式

```
hive.mapred.mode=nonstrict
```

8、mapper和reducer的个数

```
不是mapper和redcuer个数越多越好，也不是越少越好。适合就好。

将小文件合并处理(将输入类设置为：CombineTextInputFormat)
通过配置将小文件合并:
mapred.max.split.size=256000000
mapred.min.split.size.per.node=1
mapred.min.split.size.per.rack=1
hive.input.format=org.apache.hadoop.hive.ql.io.CombineHiveInputFormat
```

```
手动设置：
set mapred.map.tasks=2;

reducer的个数(自动决定和手动设置)：
mapred.reduce.tasks=-1
hive.exec.reducers.max=1009
```

9、配置jvm重用

```
//jvm rask数量
mapreduce.job.jvm.numtasks=1;

//允许重用的task
set mapred.job.reuse.jvm.num.tasks=1;
```

10、索引是一种hive的优化：(索引并不好)

11、分区本身就是hive的一种优化：

12、job的数量：

一般是一个查询产生一个job，然后通常情况一个job，可以是一个子查询、一个join、一个group by 、一个limit等一些操作。

```
1个job:
select
t1.*
from t_user1 t1
left join t_user2 t2
on t1.id = t2.id
where t2.id is null
;

如下3个job:
select
t1.*
from t_user1 t1
where id in (
select
t2.id
from t_user2 t2
limit 1
)
;
```

13、analyze

```
参考官网:https://cwiki.apache.org/confluence/display/Hive/StatsDev

Analyze，分析表（也称为计算统计信息）是一种内置的Hive操作，可以执行该操作来收集表上的元数据信息。这可以极
```

大的改善表上的查询时间，因为它收集构成表中数据的行计数，文件计数和文件大小（字节），并在执行之前将其提供给查询计划程序。

已经存在表的Analyze语法：

```
ANALYZE TABLE [db_name.]tablename [PARTITION(partcol1[=val1], partcol2[=val2], ...)]  --
(Note: Fully support qualified table name since Hive 1.2.0, see HIVE-10007.)
  COMPUTE STATISTICS
  [FOR COLUMNS]          -- (Note: Hive 0.10.0 and later.)
  [CACHE METADATA]       -- (Note: Hive 2.1.0 and later.)
  [NOSCAN];
```

例1(指定分区)、

```
ANALYZE table dw_employee_hive partition(bdp_day=20190701) COMPUTE STATISTICS;
```

收集表的bdp_day=20190701的这个分区下的所有列列相关信息。它是一个细粒度的分析语句。它收集指定的分区上的元数据，并将该信息存储在Hive Metastore中已进行查询优化。该信息包括每列，不同值的数量，NULL值的数量，列的平均大小，平均值或列中所有值的总和（如果类型为数字）和值的百分数。

例2(指定所有列)、

```
ANALYZE table dw_employee_hive partition(bdp_day=20190701) COMPUTE STATISTICS FOR
COLUMNS;
```

收集表的bdp_day=20190701的这个分区下的所有列相关信息。

例3(指定某列)、

```
ANALYZE table dw_employee_hive partition(bdp_day=20190701) COMPUTE STATISTICS FOR COLUMNS
snum,dept;
```

例4、

```
ANALYZE TABLE dw_employee_hive partition(bdp_day=20190701) COMPUTE STATISTICS NOSCAN;
```

收集指定分区相关信息，然后不进行扫描。

测试分析后的结果。
例1、

```
DESCRIBE EXTENDED dw_employee_hive partition(bdp_day=20190701);
```

描述结果：

```
...parameters:{totalSize=10202043, numRows=33102, rawDataSize=430326, ...
```

例2、

```
desc formatted dw_employee_hive partition(bdp_day=20190701) name;
```

结果如下：

```
col_name  data_type   min max num_nulls   distinct_count  avg_col_len max_col_len
num_trues   num_falses  comment

name string 0 37199 4.0 4 from deserializer
```

注意：
对分区表的分析，一般都要指定分区，如对全表分析，则可以这样使用partition(bdp_day).
优化后查询结果可以参考:https://www.cnblogs.com/lunatic-cto/p/10988342.html

十五、hive的存储过程

```
--创建 vi hp.hql 加入以下内容
CREATE FUNCTION hello(text STRING)
 RETURNS STRING
BEGIN
 RETURN 'Hello, ' || text || '!';
END;

FOR item IN (
SELECT id,name FROM test
)
LOOP
      PRINT item.id || '|' || hello(item.name);
END LOOP;
-------------------------------------------------------------------------
--创建vi hp1.hql 加入以下内容

CREATE PROCEDURE set_message(IN name STRING, OUT result STRING)
BEGIN
 SET result = 'Hello, ' || name || '!';
END;

-- Now call the procedure and print the results
DECLARE str STRING;
CALL set_message('world', str);
PRINT str;
-------------------------------------------------------------------------
                              测试include
--创建hp2.hql

use qf24;
create procedure select_u5()
begin
select * from qf24.u5;
end;


create function hello(text string)
returnS string
BEGIN
RETRUEN 'Hello,' || text || '!';
END;

create procedure select_u53()
begin
FOR item IN(
SELECT id,name FROM qf24.u5 limit 3
)
loop
        println item.id || '|' || item.name || '|' || hello(item.name);
end loop;
end;
```

```
create procedure pc()
begin
DECLARE tabname VARCHAR DEFAULT 'qf24.u5';
DECLARE id INT;
DECLARE cur CURSOR FOR 'SELECT id FROM ' || tabname;
OPEN cur;
FETCH cur INTO id;
WHILE SQLCODE=0 THEN
  PRINT id;
  FETCH cur INTO id;
END WHILE;
CLOSE cur;
end;

--创建hp2imp.hql
--加入以下语句

include /usr/local/sc/hp2.hql

call select_u5();

call select_u53();

call hello("text");

call pc();
```

十六、hplsql

hive 2.x版本默认就有

hive 1.x 配置hplsql过程请见hive的存储过程文档。

十七、hive的LZO压缩

请见文档。