Hive

Hive Wiki

Apache Hive

Apache Hive是数据仓库软件，查询分布式存储系统的大数据集，提供以下工具：

- 可以进行简单的数据ETL
- 将各种格式的数据结构化
- 可以访问HDFS的文件，也可以处理HBase的数据
- 通过MR来查询数据

Hive可以和Hadoop、HBase整合，下面是Hive和HBase整合的使用方法，如果Hbase是单节点，则可以使用Hive Cli命令访问Hbase：

```
$HIVE_SRC/build/dist/bin/hive --auxpath $HIVE_SRC/build/dist/lib/hive-hbase-handler-0.9.0.jar,$HIVE_SRC/build/dist/lib/hbase-0.92.0.jar,$HIVE_SRC/build/dist/lib/zookeeper-3.3.4.jar,$HIVE_SRC/build/dist/lib/guava-r09.jar --hiveconf hbase.master=hbase.yoyodyne.com:60000
```

如果HBase使用分布式集群，三个ZK的quorum 机器，则用下列命令：

```
$HIVE_SRC/build/dist/bin/hive --auxpath \$HIVE_SRC/build/dist/lib/hive-hbase-handler-0.9.0.jar,$HIVE_SRC/build/dist/lib/hbase-0.92.0.jar,$HIVE_SRC/build/dist/lib/zookeeper-3.3.4.jar,$HIVE_SRC/build/dist/lib/guava-r09.jar --hiveconf hbase.zookeeper.quorum=zk1.yoyodyne.com,zk2.yoyodyne.com,zk3.yoyodyne.com
```

详见HBase Integration

Hive Programming总结

Introduce

Hive提供了SQL查询来查找存储于hadoop的数据——HiveQL（Hive Query Language）
Metastore是一个单独的关系型数据库，用来存储hive的元数据
相比于Hive，HBase提供了行级的更新（row update）
`hive.metastore.warehouse.dir` 存储hive数据的目录

CLI

`hive --define name=value`可以用这个给hive的cli里面传参，参数名为`name`，值为`value`
`--define`等价于`--hivevar`
`hive --hiveconf name=value`可以用这个来设置hive的配置参数（也可以是其他值，一般是配置）（define、hivevar、hiveconf都是变量的不同命名空间，而不同命名空间起到的作用不同，例如hiveconf对应的就是hive的配置参数）
`hive -e "select * from tb"` 参数`-e`直接跟查询语句，`-S`以silent方式将结果展示——不显示Ok、耗时等说明信息
`hive -f file.hql` 执行hiveql脚本，或者在命令行内执行`source file.hql`
貌似新版的hive中没有src表了，可以自己创建一个dual来作为默认的测试表
`.hiverc`在hive执行之前执行，可以用于配置参数（语法还是hive的语法），也可以用`hive -i file`来指定文件

> hive 提供自动缩进
> hive 提供直接执行shell命令，例如：`!pwd;`
> hive 直接执行hadoop命令，例如：`dfs -ls /;`
> hive 注释，类似oracle ——`--`

Programming

Data Type

Base type

Type	Size	Example
tinyint	1 byte signed integer	11
smallint	2byte signed integer	20
int	4 byte signed integer	20
bigint	8 byte signed integer	20
boolean	boolean true or false	true
float	single precision	3.14159

	floating point	
double	double precision ..	3.14159
string	sequence of characters	‘this is a man’
timestamp	integer,float,string	1327882394 (Unix epoch seconds),1327882394.123456789 (Unix epochseconds plus nanoseconds), and ‘2012-02-0312:34:56.123456789’ (JDBCcompliantjava.sql.Timestampformat)
binary	array of bytes	

这些类型都是会用*Java*实现的 `cast(s as float)` cast来进行类型转换，从小的类型转到大的类型

Collection type

Type	Description	Example
struct	like C,字段通过`.`来获得	struct(‘John’, ‘Doe’)
map	使用数据型获得[‘key’],map是成对出现，第一个作为key，第二个作为value	map(‘first’, ‘John’, ‘last’, ‘Doe’)
array	通过[index]来获得值	array(‘John’,’Doe’)

将这些collection字段嵌入到表类型中，可以减少磁盘查找（因为减少了外键关联），所以提升了速度

用于创建表结构

```
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 terminatedby '\001'
collection items terminated by '\002'
map keys terminated by '\n'
stored as textfile
;
```

默认的文件分隔符：

- \n 每一行表示一个record
- $^{A}(controlA)$ 区分feilds，字段分隔符，octal code为 `\001`
- ^{B} 区分元素是在一个array、struct或map中，octal code为 `\002`
- ^{C} 分开key和value，在map中， `\003`

数据示例：

```
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^AFederal Taxes^C.15^BState
Taxes^C.03^BInsurance^C.
1^A200 Chicago Ave.^BOak Park^BIL^B60700
Bill King^A60000.0^AFederal Taxes^C.15^BState
Taxes^C.03^BInsurance^C.
1^A300 Obscure Dr.^BObscuria^BIL^B60100
```

相对应的Json数据：

```
{
"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
}
}
```

但是奇怪的是为什么`\033`这种说是非单字符呢？

传统的数据库，数据库文件只有数据库能操作，而hive的数据库文件可以用任何方式修改。这就叫：*schemaonread*

DDL && DML

DDL

HiveQL是Hive Query Language，不完全满足于现在任何的ANSI SQL 标准。
HiveQL不提供行insert/update/delete，也不提供事务处理
Hive在数据库概念上仅仅是一个表目录（Catalog or namespace of tables）
SQL语法有点类似Mysql，没有显式的用`use database`的话，则默认是`default`数据库

```
create database if not exists db_name
location '/locate/...'
comment 'the comment of the db'
with dbproperties('creator'='weill')
```

创建数据库，并指定位置，否则位置就是配置文件中的，下面为数据库建一个目录
`describe database db_name` 查看数据库comment
`describe database extended db_name` 查看dbproperties
`drop database if exists db_name cascade;` 后面加上cascade，即便是数据库中有表也已删除
`alter database db_name set dbproperties('key'='value')` 增加dbproperties
创建表：

```
create table if not exists db.table(
name string comment 'name',
address struct<street:string,city:string> comment 'address',
ded map<string,float>,
sub array< string>
)comment 'description of the table '
tblproperties('creator'='weill')
location 'somplace'
```

这里指定了位置，则把文件传入到这个目录，直接就可以查询数据了（如果使用的默认的文件格式textfile）

```
create table if not exists tb_name like some_table
```

describe可以查看到列，用法和db一样
ExternalTable 如果表的数据和外部共享，则可以创建时`create external table`，这样在drop表的时候，数据还是存在的

```
create table if not exists db_name(...)partitioned by (country string,
state string)
```

创建表，并且按照后面的字段分区，并且分区字段可以不属于表里面的列（一般也不用在表里面再包含这些列了，浪费空间，因为这些列的数据已经被当做目录名存在了），如上面，目录可能是：

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

这里有个问题：按理指定分隔符、然后文件传入指定目录即可，就可以查询了，但是如果是分区表，数据要重新整理，这个呢，重新创建数据？这种情况是否没法是external table
Ans：分区也可以指定为external table，地址也可以指定
例如：

```
alter table log_messages add partition(year=2012, month=1)
location 'hdfs://master_server/data/log_messages/2012/01/02'
```

`show partitions db_name [partition(country='us')]` 可以不指定某个partition，显示所有partition的信息，也可以指定某个，展示这个下面的其他partition的信息
创建语句后跟上 `stored as textfile[sequencefile|rcfile]` 等，可以设置存储格式，rcfile等格式是做了IO优化和压缩的，所以性能可能会更好。textfile默认就是按照行做记录区分，其实record编码是input format决定的（Java实现–org.apache.hadoop.mapred.TextInputFormat）。record被解析称作：序列化和反序列化（serializer/deserializer or SerDer for short），SerDer使用另一个Java类：`org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe`
对应也有Output format，对于TextFile格式，对应的类是：`org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat`
也可以指定第三方的input、output format。例如：下面使用avro的格式（row format），然后指定解析方式（input、out format）：

```
CREATE TABLE kst
PARTITIONED BY (ds string)
ROW FORMAT SERDE 'com.linkedin.haivvreo.AvroSerDe'
WITH SERDEPROPERTIES ('schema.url'='http://schema_provider/kst.avsc')
STORED AS
INPUTFORMAT 'com.linkedin.haivvreo.AvroContainerInputFormat'
OUTPUTFORMAT 'com.linkedin.haivvreo.AvroContainerOutputFormat';
```

```
alter table db_name change column column_old_name column_new_name type
alter table db_name add columns (app_name string comment '');
```

`alter table name_old rename to name_new` 重命名

```
alter table tb_name add if not exits
partition(year=2011, month=1, day=1) location='/logs/2011/1/1'
alter table tb_name partition(year=2011, month=1, day=1) set
location='/logs/2011/1/1'
alter table tb_name drop if exists partition(year=2011, month=1, day=1)
alter table db_name change column old_column_name new_column_name
column_type comment 'something about the column' after some_column
```

after语句可以调整column的位置（如果想放在第一个，则用first替换after some_column即可）
`alter table db_name replace columns(a_column int comment 'dosomting');` 替换所有已存在的列为下面指定的列

```
alter table tb_name partition(...) set fileformat seqencefile;
```

DML

`load data local path 'somepath' [override] into table tb_name [partition(field=value)]` partition这个只对分区表有效（如果local指定，则将数据从local拷贝到DFS上的最终目录–/hive/warehouse/…，否则将DFS上的移动到最终目录）
`insert (override|into) table tb_name partition(field=value) select * from tb_name where...` 如果一个表的数据要插入到多个表则用下面的语句效率更高，因为只对源表做一次扫描：

```
from source_tbname st
insert override table dst_a partition(field=valuea)
select * where st.field=somevalue
insert override table dst_b partition(field=valueb)
select * where st.field=somevalue;
```

动态分区插入，一般插入数据像上面的，需要指定partition对应的field和value，Hive可以自动匹配，需要设置参数 `hive.exec.dynamic.partition.mode=nonstrict`
`insert into table tb_name partition(field) select * from old_tb`
`create table new_tb as select somefields from old_tb` 将查询子集作为一个表创建
`insert overwrite local directory '/tmp/some' select * from tb_name` 将数据导出，这个也支持 `from tb_name t insert overwrite directory '/ddd' select * where`

SQL Query

对于列示array或map类型的，select的时候可以直接用索引
`select arr[0], mp['index'] from tb_name`
`select 'price.*' from tb_name`可以用正则表达式指定列（最新版本貌似没这个功能了）
`explode`可以将array、map转化成多行处理，详见*P87*
支持`limit`、`as`
```
select case
when salary < 5000 then 'low'
when salary >100000 then 'high'
else 'on' end as salary_level
from tb_name
```
一般像不带条件的查询或者条件只是partition的字段，则采用local model处理（不使用MR查询）如果想要其他操作也用local mode则配置`hive.exec.mode.local.auto=true`
hive where语句中浮点比较存在普遍的问题
`rlike`可以使用正则表达式来作为where条件
`having`在group by之后进行过滤
`select * from tb_name a join tb_nameb b on a.id = b.id where a.name='weill'`
hive假设最后关联查询中最后一个表是最大的表，所以会尽量缓冲其他表，使用流式查询最后一个表（例如上例中tb_nameb应该就是大表）
`select /*+STREAMING (s)\*/ s.name from tb_name s join tb_name b on s.id = b.id`可以使用hint来指定哪个表使用流式
`left outer join right outer join full outer join`几种关联查询
注意：关联查询时尽量使用分区字段进行数据过滤后然后再关联查询
`left semi join`类似于`a.field in (select field from b)`返回a表中on指定的字段在b表中存在的数据：
`select * from tb_name a left semi join tb_nameb on a.type=b.type);`
mapsidejoin 如果关联中一个表很小（100行），一个表很大（13亿行），而且大表中的数据有的键数值很多（键倾斜很严重），这个时候使用mapjoin就可以提升效率，mapjoin的做法是将小表存入内存，大表在map操作的时候（map reduce）就进行join，这样就避免了reduce操作（reduce操作是根据key来分配reduce任务，可能有的reduce任务就很重），从而提升效率
`select /*+ MAPJOIN(d)\*/ s.id from tbnamea s join tbnamed d a.id = b.id where s.id>1`
`order by`和传统的一样，就是对全量数据进行排序
`sort by`只对每个reducer的数据进行排序
`distribute by`控制map输出如何划分成reducer，如：
`select * from tb_name distribute by name sort by id`前面的distribute by指定每个name被放到同一个reducer中，然后每个reducer中按照id排序
`cluster by`如果同一列即作为distribute by也作为sort by，则可以直接用cluster by
`cast(salary as float)`将字段类型进行转换，而且binary类型的字段只能被转换成string，然后可以将string转换成其他的类型，以达到将binary类型转化成其他类型
`tablesample`抽样：
非分桶表抽样：

`select * from tb_name TABLESAMPLE(n rows)` 可以将n rows替换成：nM大小、0.2percent（比率）
分桶表抽样：
`select * from tb_name TABLESAMPLE(BUCKET x OUT OF y [on column])` x表示抽样的桶编号，y是桶的数量，就是按照column（可以不指定或者携程rand()随机处理）将表tb_name分成y个桶，然后取第x个桶。

HiveQL View

HiveQL View是一种逻辑结构，可以把一个查询块当做一个表来查，而避免使用大块的查询结构
有几个用途：

- 减少查询复杂度
- 使用条件限制数据（可以只局限到某些列）
- 可以将map的拆分成列，例如：`create view orders(state,city,part) as select cols['state'], cols['city'], cols['part'] from some_tb where conditions`

HiveQL indexes

Hive允许对表的某些列创建索引，某个表的索引被存在另一张表中。
index可以使用Java插件自定义设计
创建索引：
```
crate table employees(name string, salary float)
partitioned by(country string, state string);
as 'org.apache.hadoop.hive.ql.index.compat.CompatIndexHandler';
as bitmap
alter index emp_index on table employees[partition(country='us')]
rebuild;
```
更新索引
`show formated (index|indexes) on tb_name` 列出索引
`drop index if exists index_name on table tb_name`

Hive Schema

由于MR是将一个job转化成多个task，每个task 对应到一个JVM，而多个小文件每个小文件对应一个Task，这样JVM的启动和关闭会耗费很多资源，降低效率。所以不仅仅小文件对NameNode造成较大压力，而且在MR上也是不利的。（而分区其实就是生成多个小文件，所以分区的字段如果会造成多个小文件的话，会浪费很多资源）

Bucket

`create table weblog(logid int) partitioned by (dt string) clustered by (logid) 20 bukets;` 在logid上创建20个buckets，使用Hash处理，相同logid的都会分配到相同的bucket上

```
set hive.enforce.bucketing = true;
from raw_logs insert into table weblog partition(dt='2013-3-3') select * where dt='2013-3-3'
```

设置了enforce这个property，hive自动选择reducer的数量；如果不指定，则需要指定 `mapre.reduce.tasks=20(和bucket的数量相同)` 而且插入语句的select语句where条件后加上cluster by

ColummarTables

hive一般是行存储，但是也有列存储的SerDer，来使用行列混合的存储信息方式

- ***RepeatedData*** 列中存在很多重复数据
- ***ManyColumns*** 对于那种列很多，但是经常只读一部分列的表很合适

可以参考RCFile来作为存储文件格式

AlwaysUseCompression!!

Tuning-调优

HiveQL作为一种说明性语言，将SQL语句转化成MR的job，所以摸准这点，就是尽量的减少或者调优产生的MR

一个Hive job组成一个或多个Stage，Hive默认是一次执行一个Stage（也可以进行并行处理）。一个stage可能是MRjob、抽样stage、merge stage等。

Explain用来查看前面执行的语句的执行计划，产生一个抽象语法树，explain extended可以产生更多的输出

Hive因为并行，需要将查询拆分成很多MR，这些MR中至少会有一些可以并行计算，决定最佳的MR数量的因素很多，例如数据量以及对数据要做的操作

例如：有个文件大小2.8G，默认hive.exec.reducers.bytes.per.reducer大小为1G，则会分成3个reducer，而配置hive.exec.reducers.bytes.per.reducer大小为700M，则会产生4个

mapred.job.reuse.jvm.num.tasks控制JVM的重用次数（JVM重用一直保持task slots直到任务完成，所以如果某个reducer时间很长，可能会占着slot太久，空占着这个slot）

合并多个Group by操作成一个MR job，配置：hive.multigroupby.singlemr为true

Hive提供两个虚拟列：一个是拆分的文件名（INPUT_FILE_NAME)，一个是块位于文件的偏移位置(BLOCK_OFFSET_INSIDE_FILE) **都是双下划线**

推测执行（Speculative Execution）：这个是一种Hadoop的特性，由于任务分派给很多独立的机器单独执行，而由于各个节点的资源不同，可能有些任务执行就特别慢，这个时候ResourceManager可能会将任务分配个多个机器执行（拷贝数据），然后第一个执行完的，使用其结果，ResourceManager终止其他还在执行的相同拷贝的任务。达到执行上的优化（**从这点来看，hive的优化颇似MR的优化**)，参见

mapred.map/reduce.tasks.speculative.execution

###其他文件格式和压缩（File Format and Compression）
`set io.compression.codecs` 查看支持的压缩格式
`hive.exec.compress.intermediate` 配置是否启用中间压缩，即在map reduce之间的shuffle的数据是否要压缩，这个时候应该考虑压缩的算法偏向于低CPU消耗而不是高压缩率。 `mapred.map.output.compression.codec` 这个和上面对应，配置的是压缩方法，属于io.compression.codecs里面的列表中
最终是否压缩以及压缩方式：`hive.exec.compress.output`、 `mapred.output.compression.codec`

Sequence Files

压缩文件在hadoop中一般是不可拆分的，而Sequence file支持hadoop按照block拆分文件，然后按照随意的将这些block进行压缩（也就是先拆分成block，然后再按block压缩）
`create table a_seq_file_tab stored as sequencefile;`
hadoop支持按照none、record和block拆分，record为默认，这个是hadoop的配置，在mapred-site.xml中：
`mapred.output.compression.type，值为Block、record等` ——这个只对按照sequencefile压缩的文件起作用

Hive Streaming

除了一些UDF满足自定义功能外，可以类似Hadoop Streaming类似处理方式，采用外部例如shell脚本运行一些处理

P149

一些配置

- $hive.cli.print.header$ 表示打印查询结果是否打印列头信息
- $hive.mapred.mode = strict$ 表示如果是分区表，则必须有partition filter——在partition上的where语句，否则报错，也可以指定为$nonstrict$，这样就不用了
- $hive.exec.mode.local.auto$ 让Hive决定是否自动使用local mode运行
- $hive.exec.parallel$ 决定是否采取并行执行hive的stage
- $hive.exec.dynamic.partition.mode$ 控制是否动态分区（nonstrict or strict）
- $hive.exec.max.dynamic.partitions$ 动态分区允许创建的最大分区数量
- $hive.exec.max.dynamic.partitions.pernode$ 每个MR节点允许创建的最大动态分区数量（这个理解不了）
- $mapred.map/reduce.tasks.speculative.execution$ 配置推测执行，true、false

其他

InputFormat、OutputFormat是用来处理record encoding（将输入流拆分成record，或者将record格式化后给输出流）
SerDes是用来处理record parsing（将record转化成columns）
现在理解是Input是输入数据类型，输出是写入数据库的文件类型，否则还有什么东西要Output呢？
InputFormat、OutputFormat是可以设置成不同的，但是目前观测到，InputFormat格式为Text，load到hive的表中之后，warehouse中的文件查看了下，还是text类型。这里的outputformat表示的是查询结果写成文件的格式。
`Insert into table rcfile select * from textfile` 可以将textfile格式的数据转换成rcfile的格式

将InputFormat设置为Text，OutputFormat设置为RCFile，使用select，报错，因为select的表用了Text、RCFile两种存储格式都报错，所以猜测是Input到Output有问题，而且在看文件格式的时候，文件的格式是RCFile，所以OutputFormat是否就是写入Warehouse的表文件个格式（而且在使用InputFormat为RCFile，OutputFormat是Text的时候，Warehouse的文件格式是text格式的，所以有理由相信这里的Outputformat就是写入Warehouse的数据的文件格式）