Hive教程

Hive概念

Hive是什么

Hive是基于Apache Hadoop的数据仓库，对于数据存储于处理，Hadoop提供了主要的扩展和容错能力。

Hive设计的初衷是：对于大量数据，是的数据汇总，查询和分析更加简单，它提供了SQL，允许用户更加简单的进行查询，汇总和数据分析，同时，Hive的SQL给予用户多种方式来集成自己的功能，然后做可定制化的查询。

Hive不适合做什么

Hive不是为在线事务处理而设计，它最适合用于传统的数据仓库任务

数据单元

根据颗粒度的顺序，Hive数据被组织成为：

- 数据库：命名空间功能，为了避免表，视图，分区，列等等的命名冲突，数据库也可以用于加强用户或用户组的安全
- 表：相同数据库的同类数据单元
- 分区：每个表可以有一个或多个用于决定数据如何存储的分区键。分区（除存储单元外）也允许用户有效的识别满足指定条件的行；（用户需要保证分区名字和数据内容之间的关系，分区列是虚拟列，他们不是数据本身的一部分，而是源于数据加载）
- 桶（Buckets or Clusters）:每个分区的数据，基于表的一些列的哈希函数值，又被分割成桶，不同于分区列，这些桶可以被用于有效的抽样数据。

数据类型

- 整形
  - TINYINT：1个字节的整型
  - SMAKKINT：2个字节的整型
  - INT：4个字节的整型
  - BIGINT：8个字节的整型
- 布尔类型
  - BOOLEAN：TRUE / FALSE
- 浮点数
  - FLOAT：单精度浮点数
  - DOUBLE：双精度浮点数
- 定点数
  - DECIMAL：用户可以指定范围和小数点位数
- 字符串
  - STRING：在特定的字符集中的额一个字符串序列
  - VARCHAR：在特定的字符集中的一个有最大长度限制的字符串序列
  - CHAR：在特定的字符集中的一个指定长度的字符串序列
- 日期和时间
  - TIMESTAMP：一个特定的时间点，精确到纳秒（时间戳）
  - DATE：日期
- 二进制
  - BINARY：一个二进制位序列
- 复杂类型
  - ARRAY：有序同类型的集合
  - MAP：key-value，key必须为原始类型，value可以为任意类型
  - STRUCT：字段集合，类型可以不同
    - UNION：在有限的取值范围内的一个值

内置运算和函数

参考：https://blog.csdn.net/yimingsilence/article/details/76529473

SQL

- 可以使用**WHERE** 从表中进行筛选
- 可以使用**SELECT** 从表中查询指定的列
- 两个表之间可以**JOIN**
- 可以在多个**GROUP BY** 的列上使用聚合
- 可以存储查询结构到另一个表
- 可以下载表的内容到本地目录
- 可以存储查询结果到Hadoop的分布式文件系统目录上
- 可以管理表和分区（创建，删除，和修改）
- 可以使用自定义脚本，来定制Map/Reduce作业

用法和实例

创建表

```
CREATE TABLE page_view(
  viewTime INT,
  userid BIGINT,
  page_url STRING,
  referrer_url STRING,
  ip STRING COMMENT 'IP Address of the User'
)
COMMENT 'This is the page view table'
PARTITIONED BY(
  dt STRING,
  country STRING
)
STORED AS SEQUENCEFILE;
```

解释：表的列被指定相应的类型，这些都和SQL类似，备注（COMMENT）可以基于列的级别，也可以基于表的级别，另外PARTITIONED关键词定义的分区列与数据列是不同的，分区列实际上不存储数据，当使用这种方式创建爱你表的时候，我们假设数据文件的内容，字段之间以ASSCII 001分隔，行之间以换行符分隔。当然我们也可以指定分隔符，指定分隔符例子：

```
CREATE TABLE page_view(
  viewTime INT,
  userid BIGINT,
  page_url STRING,
  referrer_url STRING,
  ip STRING COMMENT 'IP Address of the User'
)
COMMENT 'This is the page view table'
PARTITIONED BY(
  dt STRING,
  country STRING
)
-- ROW FORMAT语句指定分隔符
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '\t'
STORED AS SEQUENCEFILE;
```

STORED AS SEQUENCEFILE 表示这个数据是以二进制格式进行存储数据在hdfs上

对表的指定列进行分桶是一个不错的选择，他可以有效的对数据集进行抽样查询，如果没有分桶，则会进行随机抽样，由于在查询的时候，需要扫描所有的数据，因此，效率不高，下面的例子描述了在表page_view的uerid列上进行分桶的例子：

```
CREATE TABLE page_view(
  viewTIME INT,
  userid BIGINT,
  page_url STRING,
  referrer_url STRING,
  ip STRING COMMENT 'IP Address of the User'
)
COMMENT 'This is the page view table'
PARTITIONED BY(
  dt STRING,
  country STRING
)
CLUSTERED BY (userid)
SORTED BY (viewTime)
INTO 32 BUCKETS
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '1'
    COLLECTION ITEMS TERMINATED BY '2'
    MAP KEYS TERMINATED BY '3'
STORED AS SEQUENCEFILE;
```

通过一个userid的哈希函数，表被分成32个桶，在每个桶中的数据，是以viewTime升序进行存储，这样组织数据允许用户有效地在这n个桶上进行抽样，合适的排序使得内部操作充分利用熟悉的数据结构来进行更加有效的查询。指定了字段是如何分隔的，集合项（数组和map）如何分隔的，以及Map的key是如何分隔的。

浏览表和分区

```
SHOW TABLES;
SHOW TABLES 'page.*';  -- 正则表达式
SHOW PARTITIONS page_view;  -- 列出表的分区，没有分区就抛出错误
DESCRIBE page_view;  -- 列出表的列和列类型
DESCRIBE EXTENDED page_view; -- 表的信息，常用于调试
DESCRIBE EXTENDED page_view PARTITION (dt='2018-6-25')  -- 分区信息， 常用于调试
```

修改表

```
ALTER TABLE old_table_name RENAME TO new_table_name; -- 重命名
ALTER TABLE old_table_name REPLACE COLUMNS(col1 TYPE, ......) -- 对列进行重命名
ALTER TABLE tab1 ADD COLUMNS (c1 INT COMMENT 'a new int column', c2 STRING DEFAULT 'def val');
```

删除表

```
DROP TABLE pv_users;
ALTER TABLE pv_users DROP PARTITION(ds='2018-6-26') -- 删除分区
```

加载数据

要加载数据到Hive表有许多种方式。用户可以创建一个"外部表"来指向一个特定的HDFS路径，用这种方法，用户可以使用HDFS put或者copy命令，复制一个文件到指定的位置，并且附上相应的行格式信息创建一个表指定这个位置，一旦完成，用户就可以转换数据和插入他们到任意的Hive表中了，例子：文件/tmp/pv_2018-06-26.txt包含都好分隔的页面访问记录。这需要以合适的分区加载到表page_view

```
CREATE EXTERNAL TABLE page_view_stg(
  viewTime INT,
  userid BIGINT,
  ip STRING COMMENT 'IP Address of the User',
  country STRING COMMENT 'country of origination'
)
COMMENT 'This is the staging page view table'
ROW FORMAT DELIMITED FIELDS TERMINATED BY '44' LINES TERMINATED BY '12'
STORED AS TEXTFIKE
LOCATION '/user/data/staging/page_view'

hadoop dfs -put /tmp/pv_2018-06-26.txt /user/data/staging/page_view

FROM page_view_stg pvs
INSERT OVERWRITE TABLE page_view PARTITION(dt='2018-06-26', country='CN')
SELECT pvs.viewTime, pvs.userid, pvs.page_url, pvs.referrer_url, null, null, pvs.ip
WHERE pvs.country = 'CN'
```

其中 '44'是逗号的ASCII码，'12'是换页符，null是作为目标表中的数组和map类型插入，如果指定了合适的行格式，这些值也可以来自外部表。

如果在HDFS上有一些历史数据，用户想增加一些元数据，以便于可以使用Hive来查询和操纵这些数据，就可以使用这种方法。

另外，系统也支持直接从本地文件系统上加载数据到Hive表，表的格式与输入文件的格式需要相同，如果按文件/tmp/pv_2018-06-26包含了CN数据，然后我们不需要像前面例子那样的筛选，我们可以使用以下语法进行加载数据：

```
LOAD DATA LOCAL INPATH /tmp/pv_2018-06-26.txt INTO TABLE page_view PARTITION(date='2018-06-26',
country='CN')
```

路径参数可以是一个目录，一个文件，一个通配符，如果参数是目录，它不能包含子目录

在加载文件非常大的情况下，用户可以采用并行加载数据的方式（使用Hive的外部工具），只要文件在HDFS上，就可以执行以下语句：

```
LOAD DATA INPATH '/user/data/pv_2018-06-26.txt' INTO TABLE page_view PARTITION(date='2018-06-
26', country='CN')
```

查询和插入数据

简单的查询

```
SELECT user.*
FROM user
WHERE user.active = 1;
```

基于查询的分区

在一个查询中，要使用什么分区，是由系统根据where的分区列上条件自动的决定，例子：

```
SELECT page_views.*
FROM page_views
WHERE page_views.date >= '2008-03-01'
```

注意：在这里使用的page_views.date是用PARTITIONED BY(date DATATIME, country STRING) 定义的，根据自己的分区命名的不同做修改

连接

```
SELECT pv.*, u.gender, u.age
FROM user u JOIN page_view pv ON(pv.userid = u.id)
WHERE pv.date = '2018-6-26';
```

想实现外连接，用户可以使用LEFT OUTER， RIGHT OUTER或者FULL OUTER关键词来指定不同的外连接，例子：

```
SELECT pv.*, u.gender, u.age
FROM user u FULL OUTER JOIN page_view pv ON (pv.userid = u.id)
WHERE pv.date = '2018-06-26';
```

为了检查key在另外一个表中是否存在，用户可以使用LEFT SEMI JOIN，例子：

```
SELECT u.*
FROM user u LEFT SEMI JOIN page_view pv ON (pv.userid = u.id)
WHERE pv.date = '2018-06-26'
```

为了连接多个表，可以使用以下语法：

```
SELECT pv.*, u.gender, u.age, f.friends
FRO< page_view pv JOIN user u ON(pv.userid = u.id) JOIN friend_list f ON(u.id=f.uid)
WHERE pv.date='2018-6-26'
```

注意：把最大的表放在join的最右边，可以得到很好的性能

聚合

统计用户每个性别的人数，例子：

```
SELECT pv_users.gender, COUNT(DISTINCT pv_users.userid)
FROM pv_users
GROUP BY pv_users.gender
```

可以同时做多个聚合，然而两个聚合函数不能同时使用DISTINCT作用于不同的列，例子：

```
SELECT pv_users.gender, COUNT(DISTINCT pv_users.userid), COUNT(*), SUM(DISTINCT pv_users.userid)
FROM pv_users
GROUP BY pv_users.gender;
```

但是以下情况是不被允许的：

```
INSERT OVERWRITE TABLE pv_gender_agg
SELECT pv_users.gender, count(DISTINCT pv_users.userid), count(DISTINCT pv_users.ip)
FROM pv_users
GROUP BY pv_users.gender;
```

多表/文件插入

聚合或简单查询的输出可以插入到多个表中，或者甚至是HDFS文件：

```
FROM pv_users
INSERT OVERWRITE TABLE pv_gender_sum
SELECT pv_users.gender, count_distinct(pv_users.userid)
GROUP BY pv_users.gender

INSERT OVERWRITE DIRECTORY '/user/data/tmp/pv_age_sum'
SELECT pv_users.age, count_distinct(pv_users.userid)
GROUP BY pv_users.age;
```

第一个插入语句将结果插入到Hive表中，第二个插入语句是将结果写到HDFS文件

动态分区插入

加载到多个分区

```
FROM page_view_stg pvs
INSERT OVERWRITE TABLE page_view PARTITION(dt='2018-6-26', country='CN')
    SELECT pvs.viewTime, pvs.userid, pvs.page_url, pvs.referrer_url, null, null, pvs.ip WHERE
pvs.country = 'CN'
INSERT OVERWRITE TABLE page_view PARTITION(dt='2018-06-26', country='CN')
    SELECT pvs.viewTime, pvs.userid, pvs.page_url, pvs.referrer_url, null, null, pvs.ip WHERE
pvs.country = 'CA'
INSERT OVERWRITE TABLE page_view PARTITION(dt='2018-06-26', country='UK')
    SELECT pvs.viewTime, pvs.userid, pvs.page_url, pvs.referrer_url, null, null, pvs.ip WHERE
pvs.country = 'UK';
```

为了加载数据到全部的country分区到指定的日期，我们必须在输入数据汇总为每个country增加一条插入语句，这是非常不方便的，因为我们需要提前创建且知道已经存在哪些country分区列表，如果哪天这些country列表变了，我们必须修改我们的插入语句，也应该创建相应的分区，这是非常低效的，因为每个语句都传化成一个MapReduce作业。

动态分区插入（Dynamic partition insert）就是为了解决以上问题而设计的，他通过动态的决定在扫描数据的时候，哪些分区应该创建和填充，在动态分区插入中，输入列被评估，这行应该插入到哪个分区，如果分区没有创建，他将自动创建这个分区，使用这个特征，我们仅仅需要将插入语句来创建和填充所有需要的分区，另外因为只有一个插入语句，相应的也只有一个MapReduce作业，会显著提高性能且降低Hadoop的集群负载

```
FROM page_view_stg pvs
INSERT OVERWRITE TABLE page_view PARTITION(de='2018-06-26', country)
    SELECT pvs.viewTime, pvs.userid, pvs.page_url, pvs.referrer_url, null, null, pvs.ip,
pvs.country;
```

与多条插入语句对比：

- country出现在PARTITION后面，但是没有具体的值，这种情况，country就是一个动态的分区列，另一方面，dt有一个值，就意味着他是一个静态的分区列。如果一个列是动态分区列，他的值将会使用输入列的值。
- 一个额外的pvs.country列被加入在查询语句中，这对动态分区列来说，相当于输入列，注意，对静态分区列，我们不需要添加一个输入列，因为在PARTITION语句中，他的值已经知道。注意：动态分区列的值查出来是有序的，且是放在select语句最后

插入到本地文件

```
INSERT OVERWRITE LOCAL DIRECTORY '/tmp/pv_gender_sum'
SELECT pv_gender_sum.*
FROM pv_gender_sum;
```

抽样

抽样语句允许用户对数据抽样查询，而不是全表查询，当前，抽样是对那些在CREATE TABLE语句的CLUSTERED BY修饰的列上，例子，选择第三个桶：

```
INSERT OVERWRITE TABLE pv_gender_sum_sample
SELECT pv_gender_sum.*
FROM pv_gender_sum TABLESAMPLE(BUCKET 3 OUT OF 32);
```

通常TABLESAMPLE的语法是这样：

```
TABLESAMPLe (BUCKET x OUT OF y)
```

y必须是桶数量的因子或倍数，桶的数量是在创建表的时候指定的，抽样所选的桶由桶大小，y和x共同决定，如果y和桶大小相等，则抽样所选取的桶是x对y的求模结果

```
TABLESAMPLE(BUCKET 3 OUT OF 16)
```

这将抽样第3个和第19个桶。桶的编号从0开始。

```
TABLESAMPLE(BUCKET 3 OUT OF 64 ON userid)
```

这将抽取第3个桶的一半。

全连接

支持`union all`，如果假设我们有两个不同的表，分别用来记录用户发布的视频和用户发布的评论，以下例子是一个union all 的结果与用户表再连接的查询：

```
INSERT OVERWRITE TABLE actions_users
SELECT u.id, actions.date
FROM (
    SELECT av.uid AS uid
    FROM action_video av
    WHERE av.date = '2008-06-03'

    UNION ALL

    SELECT ac.uid AS uid
    FROM action_comment ac
    WHERE ac.date = '2008-06-03'
    ) actions JOIN users u ON(u.id = actions.uid);
```

数组操作

```
CREATE TABLE array_table (int_array_column ARRAY<INT>);
```

假设pv.friends 是类型`ARRAY<INT>`（也就是一个整型数组），用户可以通过索引号获取数组中特定的元素，如下：

```
SELECT pv.friends[2]
FROM page_views pv;
```

这个查询得到的是pv.friends里的第三个元素。

用户也可以使用函数`size`来获取数组的长度，如下：

```
SELECT pv.userid, size(pv.friends)
FROM page_view pv;
```

Map（关联数组）操作

Map提供了类似于关联数组的集合。这样的结构不仅可以由程序创建。我们也将很快可以继承这个。假设pv.properties是类型map<String,String>，如下：

```
INSERT OVERWRITE page_views_map
SELECT pv.userid, pv.properties['page type']
FROM page_views pv;
```

这将查询表`page_views`的'page_type'属性。

与数组相似，也可以使用函数`size`来获取map的大小：

```
SELECT size(pv.properties)
FROM page_view pv;
```

定制Map/Reduce脚本

通过使用Hive语言原生支持的特征，用户可以插入他们自己定制的mapper和reducer在数据流中。例如，要运行一个定制的mapper脚本script-map_script和reducer脚本script-reduce_script)，用户可以执行以下命令，使用TRANSFORM来嵌入mapper和reducer脚本。

注意：在执行用户脚本之前，表的列会转换成字符串，且由TAB分隔，用户脚本的标准输出将会被作为以TAB分隔的字符串列。用户脚本可以输出调试信息到标准错误输出，这个信息也将显示hadoop的详细任务页面上。

```
FROM (
     FROM pv_users
     MAP pv_users.userid, pv_users.date
     USING 'map_script'
     AS dt, uid
     CLUSTER BY dt) map_output

 INSERT OVERWRITE TABLE pv_users_reduced
     REDUCE map_output.dt, map_output.uid
     USING 'reduce_script'
     AS date, count;
```

map脚本样本（weekday_mapper.py)

```
import sys
import datetime

for line in sys.stdin:
  line = line.strip()
  userid, unixtime = line.split('\t')
  weekday = datetime.datetime.fromtimestamp(float(unixtime)).isoweekday()
  print ','.join([userid, str(weekday)])
```

当然，对于那些常见的select转换，MAP和REDUCE都是“语法糖”。内部查询也可以写成这样：

```
SELECT TRANSFORM(pv_users.userid, pv_users.date) USING 'map_script' AS dt, uid CLUSTER BY dt
FROM pv_users;
```

Co-Groups

在使用map/reduce的群体中，cogroup是相当常见的操作，它是将来自多个表的数据发送到一个定制的reducer，使得行由表的指定列的值进行分组。在Hive的查询语言中，可以使用以下方式，通过使用union all和cluster by来实现此功能。假设我们想对来自表actions_video和action_comment的行对uid列进行分组，且需要发送他们到reducer_script定制的reducer，可以使用以下语法：

```
FROM (
```

```
    FROM (
            FROM action_video av
            SELECT av.uid AS uid, av.id AS id, av.date AS date

           UNION ALL

            FROM action_comment ac
            SELECT ac.uid AS uid, ac.id AS id, ac.date AS date
    ) union_actions
    SELECT union_actions.uid, union_actions.id, union_actions.date
    CLUSTER BY union_actions.uid) map

 INSERT OVERWRITE TABLE actions_reduced
    SELECT TRANSFORM(map.uid, map.id, map.date) USING 'reduce_script' AS (uid, id,
reduced_val);
```

参考链接：

https://legacy.gitbook.com/book/strongyoung/hive/details

https://blog.csdn.net/bingduanlbd/article/details/52075058