DROP TABLE IF EXISTS employee ;

CREATE TABLE IF NOT EXISTS employee

(

id INT COMMENT 'ID'

,name STRING COMMENT '姓名'

,age INT COMMENT '年龄'

,phone STRING COMMENT '电话'

)

PARTITIONED BY -

(

dt STRING COMMENT '日期'

)

LIFECYCLE 31

;

这里介绍两个概念：

### partition by

1、在Hive Select查询中一般会扫描整个表内容，会消耗很多时间做没必要的工作。有时候只需要扫描表中关心的一部分数据，因此建表时引入了partition概念。

2、分区表指的是在创建表时指定的partition的分区空间。

3、如果需要创建有分区的表，需要在create表的时候调用可选参数partitioned by，详见表创建的语法结构。

二、技术细节

1、一个表可以拥有一个或者多个分区，每个分区以文件夹的形式单独存在表文件夹的目录下。

2、表和列名不区分大小写。

3、分区是以字段的形式在表结构中存在，通过describe table命令可以查看到字段存在，但是该字段不存放实际的数据内容，仅仅是分区的表示。

4、建表的语法（建分区可参见PARTITIONED BY参数）：

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]

5、分区建表分为2种，一种是单分区，也就是说在表文件夹目录下只有一级文件夹目录。另外一种是多分区，表文件夹下出现多文件夹嵌套模式。

a、单分区建表语句：create table day\_table (id int, content string) partitioned by (dt string);单分区表，按天分区，在表结构中存在id，content，dt三列。

b、双分区建表语句：create table day\_hour\_table (id int, content string) partitioned by (dt string, hour string);双分区表，按天和小时分区，在表结构中新增加了dt和hour两列。

6、添加分区表语法（表已创建，在此基础上添加分区）：

ALTER TABLE table\_name ADD partition\_spec [ LOCATION 'location1' ] partition\_spec [ LOCATION 'location2' ] ... partition\_spec: : PARTITION (partition\_col = partition\_col\_value, partition\_col = partiton\_col\_value, ...)

用户可以用 ALTER TABLE ADD PARTITION 来向一个表中增加分区。当分区名是字符串时加引号。例：

ALTER TABLE day\_table ADD PARTITION (dt='2008-08-08', hour='08') location '/path/pv1.txt' PARTITION (dt='2008-08-08', hour='09') location '/path/pv2.txt';

7、删除分区语法：

ALTER TABLE table\_name DROP partition\_spec, partition\_spec,...

用户可以用 ALTER TABLE DROP PARTITION 来删除分区。分区的元数据和数据将被一并删除。例：

ALTER TABLE day\_hour\_table DROP PARTITION (dt='2008-08-08', hour='09');

8、数据加载进分区表中语法：

LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)]

例：

LOAD DATA INPATH '/user/pv.txt' INTO TABLE day\_hour\_table PARTITION(dt='2008-08- 08', hour='08'); LOAD DATA local INPATH '/user/hua/\*' INTO TABLE day\_hour partition(dt='2010-07- 07');

当数据被加载至表中时，不会对数据进行任何转换。Load操作只是将数据复制至Hive表对应的位置。数据加载时在表下自动创建一个目录，文件存放在该分区下。

9、基于分区的查询的语句：

SELECT day\_table.\* FROM day\_table WHERE day\_table.dt>= '2008-08-08';

10、查看分区语句：

hive> show partitions day\_hour\_table;

三、总结

1、在 Hive 中，表中的一个 Partition 对应于表下的一个目录，所有的 Partition 的数据都存储在最字集的目录中。

2、**总的说来partition就是辅助查询，缩小查询范围，加快数据的检索速度和对数据按照一定的规格和条件进行管理。**

### Cyclelife

1、生命周期单位为天，取值为正整数。

2、对于非分区表，如果表数据在生命周期内没有被修改，经过指定天数后此表将会被hive自动回收（类似DROP TABLE操作）。生命周期从最后一次表数据被修改的时间起开始计算。

3、对于分区表，每个分区可以分别被回收。在生命周期内未被修改数据的分区，经过指定的天数后此分区将会被回收，否则会被保留。每个分区的生命周期是从最后一次分区数据被修改的时间起开始计算。不同于非分区表，分区表的最后一个分区被回收后，该表不会被删除。

说明

生命周期回收为每天定时启动，扫描全量分区。最后一次分区数据被修改的时间需要超过生命周期指定的时间才回收。

假设某个分区表生命周期为1天，该分区数据最后一次被修改的时间是2020年2月17日15时。如果在2020年2月18日15时之前扫描此表（不到一天），则不会回收表分区。如果2020年2月19日回收扫描时发现表分区最后一次分区数据被修改的时间超过生命周期指定的时间，则上述分区会被回收。

4、只能在表级别设置生命周期，不能在分区级设置生命周期。为分区表指定的生命周期，适用于该表所有的分区。创建表时即可指定生命周期。

5、如果没有为表指定生命周期，则表（分区）不会根据生命周期规则被自动回收。

生命周期管理功能主要是为了方便释放存储空间，简化回收数据的流程。

desc employee;

说明：desc类似于pandas的dtypes功能，可以方便的查看列名和列的属性等基本表信息

INSERT OVERWRITE TABLE employee PARTITION(dt=20200105)

VALUES (1, 'July', 18,13176890435),

(1, 'July', 18,15976890435),

(1, 'July', 18,15976890435),

(2, 'Li', 38, 13876891135),

(3, 'Zhang', 42, 18976890435),

(4, 'Wang', 25, 15076890435),

(4, 'Zhao', 25, NULL )

;

说明：

针对分区进行插入效率更高一些

DESC employee;

SELECT \*

FROM employee

WHERE dt = 20200105

limit 2

;

READ employee;

SELECT \*

FROM employee

WHERE dt = max\_pt('employee')

;

DROP TABLE if EXISTS employee\_Copy;

CREATE TABLE IF NOT EXISTS employee\_Copy AS

SELECT \*

FROM employee

WHERE name = 'July'

AND dt = max\_pt('employee')

;

READ employee\_Copy;

--简单查询

SELECT \*

FROM employee

WHERE dt = max\_pt('employee')

;

SELECT name,phone from employee

WHERE dt = max\_pt('employee');

SELECT DISTINCT name from employee

WHERE dt = max\_pt('employee');

SELECT name as 姓名 from employee

WHERE dt = max\_pt('employee');

--排序与分组

SELECT name,age from employee WHERE dt = max\_pt('employee')

ORDER by age DESC ;

SELECT name, max(age) as max\_age from employee WHERE dt = max\_pt('employee')

GROUP BY name

HAVING max\_age>19;

--条件查询

SELECT \* from employee WHERE dt = max\_pt('employee')

and name LIKE 'Ju%';

SELECT \* from employee WHERE dt = max\_pt('employee')

and phone is not NULL

and age >18 and age <=38

ORDER BY age ASC

limit 2;

read employee;

--SQL 日期函数

SELECT name,

CAST(age as string),

TO\_DATE(dt,'yyyymmdd'),

ISDATE(dt,'yyyymmdd'),

getdate(),

DATETRUNC(getdate(), 'yyyy'),

DATEDIFF(GETDATE() ,TO\_DATE(dt,'yyyymmdd'),'hh'),

DATEADD(TO\_DATE(dt,'yyyymmdd'),1,'mm'),

to\_char(DATEADD(TO\_DATE(dt,'yyyymmdd'),1,'mm'),'yyyy mm dd'),

DATEPART(DATEADD(TO\_DATE(dt,'yyyymmdd'),1,'mm'), 'yyyy'),

DATEPART(DATEADD(TO\_DATE(dt,'yyyymmdd'),1,'mm'),'mm'),

WEEKDAY(TO\_DATE(dt,'yyyymmdd'))+1

from employee WHERE dt = max\_pt('employee');

--SQL数字函数

SELECT ABS(-1.2),

ASIN(0.5),

ceil(1.1),

COALESCE(NULL, 1,2,3,12),

nvl(NULL,12),

conv('1100', 2, 10),

EXP(1),

floor(1.2),

ln(10),

log(2,64),

POW(2,3),

RAND(11),

ROUND(3.14159,3),

SQRT(9),

cbrt(27),

trunc(125.815, 2),

log2(8),

log10(100)

,sign(-2.5),

e(),

pi();

--SQL窗口函数

select MAX(age), MIN(age),AVG(age),

MEDIAN(age) ,STDDEV(age),

COUNT(\*) ,sum(age)

from employee WHERE dt = max\_pt('employee');

select name,phone,age, rank() over(partition by name order by phone) as rank ,

ROW\_NUMBER() OVER (PARTITION BY name ORDER BY age DESC) AS nums

from employee WHERE dt = max\_pt('employee');

--SQL字符串函数

select concat('ab', 'c') , length('abc'),md5('abc'),

substr("abc", 2, 1),

tolower("aBcd"),

toupper("aBcd"),

trim(' d '),

reverse('abcedfg'),

concat\_ws(':','name','july'),

replace('ababab','abab','12'),

split('a-b-c',"-"),

-- part：String 类型，支持HOST, PATH, QUERY, REF, PROTOCOL, AUTHORITY, FILE, 和 USERINFO，不区分大小写，不在此范围抛异常

parse\_url('https://ide2-cn-hangzhou.data.aliyun.com/', 'HOST');

--并集

SELECT \* FROM VALUES (1, 2), (1, 2), (3, 4) t(a, b)

UNION ALL

SELECT \* FROM VALUES (1, 2), (1, 4) t(a, b);

--带distinct

SELECT \* FROM VALUES (1, 2), (1, 2), (3, 4) t(a, b)

UNION

SELECT \* FROM VALUES (1, 2), (1, 4) t(a, b);

--交集

SELECT \* FROM VALUES (1, 2), (1, 2), (3, 4), (5, 6) t(a, b)

INTERSECT ALL

SELECT \* FROM VALUES (1, 2), (1, 2), (3, 4), (5, 7) t(a, b);

--补集

SELECT \* FROM VALUES (1, 2), (1, 2), (3, 4), (3, 4), (5, 6), (7, 8) t(a, b)

EXCEPT ALL

SELECT \* FROM VALUES (3, 4), (5, 6), (5, 6), (9, 10) t(a, b);

--JOIN

SELECT \*

FROM (

SELECT \*

FROM employee

WHERE dt = max\_pt('employee')

) a

JOIN (

SELECT \*

FROM employee\_Copy

) b

ON a.name = b.name

;

SELECT \*

FROM (

SELECT \*

FROM employee

WHERE dt = max\_pt('employee')

) a

left outer JOIN (

SELECT \*

FROM employee\_Copy

) b

ON a.name = b.name

;

SELECT \*

FROM (

SELECT \*

FROM employee

WHERE dt = max\_pt('employee')

) a

right outer JOIN (

SELECT \*

FROM employee\_Copy

) b

ON a.name = b.name

;

SELECT \*

FROM (

SELECT \*

FROM employee

WHERE dt = max\_pt('employee')

) a

full outer JOIN (

SELECT \*

FROM employee\_Copy

) b

ON a.name = b.name

;

SELECT /\*+ mapjoin(b) \*/

\*

FROM (

SELECT \*

FROM employee

WHERE dt = max\_pt('employee')

) a

JOIN (

SELECT \*

FROM employee\_Copy

) b

ON a.name = b.name

;

个人习惯通过as进行变量的指定，代码上看起来更舒服更整洁一些