Day11_Hive操作数据库

大数据-张军锋 Day11 Hive SQL

基本的表操作

创建外部表

> hdfs上的文件不会被剪切到hive表目录下，在删除表的时候源文件是不会被删除的，只有表的数据内容被删除
> 外部表相对来说更加安全些，数据组织也更加灵活，方便共享源数据。

```
create external table txt_department(
dep_id string,
dep_name string,
address string
)
row format delimited
fields terminated by '\t'
stored as textfile
```

内部表和外部表的异同

> 内部或外部表如果用load data操作，hive都会把数据剪切到hive的目录结构里
> 外部表可以通过添加location把hive表和外部目录进行映射
> 数据分析上采集到的数据(元数据)基本都是使用外部表
> 分析过程中产生的各种数据,基本都是内部表
> 加载数据源的时候尽量保持文件的格式，防止数据丢失或者失真

1. 在导入数据到外部表，数据并没有移动到自己的数据仓库目录下(如果指定了location的话)，也就是说外部表中的数据并不是由它自己来管理的！而内部表则不一样
2. 在删除内部表的时候，Hive将会把属于表的元数据和数据全部删掉；而删除外部表的时候，Hive仅仅删除外部表的元数据，数据是不会删除的！
3. 在创建内部表或外部表时加上location 的效果是一样的，只不过表目录的位置不同而已，加上partition用法也一样，只不过表目录下会有分区目录而已，load data local inpath直接把本地文件系统的数据上传到hdfs上，有location上传到location指定的位置上，没有的话上传到hive默认配置的数据仓库中。

```
create external table txt_department02(
dep_id string,
dep_name string,
address string
)
row format delimited
fields terminated by '\t'
location '/emp_dep'


create table employee_2(
  emp_id int,
  emp_name string,
  status string,
  salary double,
  status_salary double,
  in_work_date date,
  leader_id int,
  dep_id int
)
row format delimited
fields terminated by '\t'
load data local inpath '/usr/emp_dep/employee.txt' overwrite into t
able employee_2
```

drop外部表：表不存在了，数据文件依旧存在

```
drop table txt_department02
load data inpath '/emp_dep/dep.txt' overwrite into table txt_depart
ment
```

删除

```
drop table 表名;
```

根据已有表创建

表复制

```
create table dw_employee as
select cast(emp_id as int)
       ,emp_name
       ,status
       ,cast(salary as double) salary
       ,cast(status_salary as double) status_salary
       ,cast(in_work_date as date) in_work_date
       ,cast(leader_id as int) leader_id
       ,cast(dep_id as int) dep_id
from employee
```

表克隆：结构一样，不克隆数据

```
create table employee_clone like employee
select *from employee_clone
```

查看表结构

```
describe employee_clone
describe formatted employee_clone
describe extended employee_clone
```

Hive && SQL

基础练习

计算员工的 月薪, 季度薪, 年薪

```
select emp_id
       ,emp_name
       ,salary
       ,salary*3 as 3salary
       ,salary*12 as 12salary
from dw_employee
```

计算员工月收入(月薪加奖金)

```
select emp_id
       ,emp_name
       ,salary
       ,status_salary
       ,salary + status_salary  as xinshui
from dw_employee

-- 判断status_salary是否为空
select emp_id
       ,emp_name
       ,salary
       ,status_salary
       ,salary + case when  status_salary is null then 0 else statu
s_salary end as xinshui
from dw_employee

--简化操作，nvl判断是否为空
select emp_id
       ,emp_name
       ,salary
       ,status_salary
       ,salary + nvl(status_salary,0) as xinshui
from dw_employee

insert into employee_clone (emp_id,emp_name) values('ss','fff')
select *from employee_clone
```

插入一条数据id为1111，姓名为aaa的数据，其余字段空

```
-- hive 几乎不会使用insert values 的写法
insert into employee_clone (emp_id,emp_name) values('ss','fff')
select * from employee_clone
-- hive 支持delete和update，但是数据分析几乎不使用这两种方式来更新表
-- hive对这两个语法的支持需要特殊配置
delete from employee_clone
update employee set emp_id='sss'
```

查询员工表如果没有职位，显示‘普通员工’，有职位显示职位、没有入职日期显示入职日期为10-may-15,有入职日期则显示入职日期

```
select emp_id
      ,emp_name
      ,salary
      ,status_salary
      ,nvl(status,'普通员工')
      ,nvl(in_work_date,'2015_05_10')
from dw_employee
```

复制员工表,表名emp_copy

```
create table emp_copy as
select *from dw_employee
```

机构中有多少种职位（distinct）

```
-- 通过关键字distinct
select distinct status
from dw_employee

-- 通过group by进行排重
select status
from dw_employee
group by status
```

查出employee每个部门中不重复的职位(distinct)

```
select distinct dep_id
             ,status
from dw_employee
```

薪水高于6000的员工

```
select *
from dw_employee
where salary > 6000
```

职位是anaylst 的员工

```
select *
from dw_employee
where status = 'Anaylst'
```

以小写的格式展示职位信息（小写lower( ) 大写 upper ( ))

```
select emp_id
       ,emp_name
       ,lower(status)
       ,upper(status)
from dw_employee
```

忽略大小写匹配职位等于‘ANAYLST’的记录

```
select *
from dw_employee
where upper(status) = 'ANAYLST'
```

查询出薪水在5000到8000之间的员工（between and)

```
select *
from dw_employee
where salary between 5000 and 8000
```

查询出2016年入职的员工

```
select *
from dw_employee
where year(in_work_date)=2016
```

薪水不在5000到8000的员工

```
select *
from dw_employee
where salary not between 5000 and 8000
```

查询出职位是Manager或者anaylst的员工

```
select *
from dw_employee
where status='Manager' or status='Anaylst'

select *
from dw_employee
where status in ('Manager','Anaylst')
```

查询出薪水大于7000并且职位是Coder或者Anaylst的员工信息

```
select *
from dw_employee
where salary>7000
    and (status='Coder'
    or status = 'Anaylst')
```

查询出没有岗位工资的员工

```
select *
from dw_employee
where status_salary is null
```

查询有岗位工资的员工

```
select *
from dw_employee
where status_salary is not null
```

查询出不在 1001 部门和不再 1002 部门的员工

```
select *
from dw_employee
where dep_id not in (1001,1002)
```

简述DDL && DML && DCL

DDL数据定义语言

DDL(Data Definition Language)
create 数据库对象的创建
alter 修改数据库对象
drop 删除数据库对象
truncate 清空表数据，表级别的操作，删除后数据不可恢复

> **truncate和delete之间的区别**
> truncate是表级别的操作，delete是行级别的操作；truncate删除数据不能通过日志进行恢复，delete删除之后可以通过日志进行恢复

DML数据操纵语言

DML(Data Manipulation Language)
insert 插入操作
update 更新操作
delete 删除操作

将查询的结果插入到表中

```
create table dw_employee_leader like dw_employee

insert into table dw_employee_leader
select *
from dw_employee
where leader_id is null
```

insert overwrite table 先删除表中数据,再往后继续添加

```
insert overwrite table dw_employee_leader
select * from dw_employee
where leader_id is null
```

DCL 数据控制语言(基本上不使用)

Data Control Language

- 用语执行权限的授予和收回操作
- GRANT:授予，给用户授权
- Revoke:收回用户已有的权限
- Create user:创建用户
- create user usernamexxx identity by ‘123456’

排序

> **降序desc，升序asc**
> **Order by** 后面跟两个字段，主排序字段 副排序字段

查询人员信息 按薪水从高到低排序
hive的order by 的全排序是通过只设置一个reduce节点的方式来实现的

```
select *
from dw_employee
order by salary desc
```

二次排序

```
select *
from dw_employee
order by salary desc,status_salary desc
```

分组 && 聚合

聚合—-多行数据用一个函数制定的规则来进行运算
Sum：求和
Count：计数，count字段值为空的不计算,count(1)或count(*)比较推荐使用
Avg：平均值
Max：最大值
Min：最小值

分组—-为聚合创造多行数据来源的条件
Group by

> 分组和聚合一般组合起来使用

查询有多少个员工

```
select count(*)
from dw_employee
```

设置reduce数量

```
set mapreduce.job.reduces=2
```

查询参数

```
setmapreduce.job.reduces
```

count 字段值为空的不计数

```
select dep_id
      ,count(*)
from dw_employee
group by dep_id
```

查询有多少个员工姓张

```
select count(*)
from dw_employee
where emp_name like '%张%'
```

计算员工的总薪水是多少

```
select sum(salary)
from dw_employee
```

计算员工的人数总和、薪水综合、平均薪水，最高薪水、最低薪水

```
select count(*)
      ,sum(salary)
      ,avg(salary)
      ,max(salary)
      ,min(salary)
from dw_employee
```

求出每个部门的最高薪水、最低薪水、平均薪水、总薪水、总人数

```
select dep_id
      ,sum(salary)
      ,avg(salary)
      ,max(salary)
      ,min(salary)
      ,count(1)
from dw_employee
group by dep_id
```

Having 字句

求出总人数超过2人的部门的最高薪水、最低薪水、平均薪水、总薪水、总人数

```
select dep_id
      ,max(salary)
      ,min(salary)
      ,avg(salary)
      ,sum(salary)
      ,count(*)
from dw_employee
group by dep_id
having count(*)>2
```

> where是对聚合之前判断，having是聚合之后判断

子查询

查询出最高薪水的人的信息

```
select *
from dw_employee
where salary in (select max(salary) from dw_employee)
```

> in在后面添加子查询结果作为判断条件时，如果子查询中有索引的话，in是用不到这个索引的，因此in对接子查询时效率较低

```
select *
from dw_employee a
where exists(
      select maxsalary from (select max(salary) maxsalary from dw_e
mployee) b
      where b.maxsalary=a.salary
)
```

最低薪水的人

工资高于平均薪水的人的信息

```
-- 不可以使用非等判断符号
select *
from dw_employee a
where salary>(
      select avg(salary) from dw_employee
)

-- 不可以使用非等判断符号
select *
from dw_employee a
where exists(
      select avgsalary from (select avg(salary) avgsalary from dw_e
mployee) b
      where a.salary>b.avgsalary
)


-- 使用笛卡尔乘积
select a.* ,b.*
from dw_employee a
      ,(select avg(salary) avgsalary from dw_employee) b
where a.salary>b.avgsalary
```

> 如果运行出错（不让使用笛卡尔乘积），设置一下`set hive.strict.checks.cartesian.product=false;`

研发部有哪些职位

```
select distinct status
from dw_employee a
where exists(
      select dep_id from  txt_department
      where dep_name='研发部'
      and a.dep_id=dep_id
)
```

谁是妖姬的同部门的同事

> **思考：**如果有两个人叫妖姬，谁是妖姬的同事

```sql
select *
from dw_employee a
where dep_id in (
      select dep_id from dw_employee
      where map_name='妖姬'
)and a.emp_name<>'妖姬'
```

> **练习**
> 谁是艾克的下属
> 如果有两个人叫艾克，谁是艾克的下属

每个部门最高薪水的人是谁

```sql
-- hive不支持
select *
from dw_employee a
where (dep_id,salary) in(
      select dep_id
      ,max(salary) max_salary
      from dw_employee
      group by dep_id
) b

select *
from dw_employee a
where exists(
      select b.dep_id
          ,b.max_salary
      from(
          select dep_id
          ,max(salary) max_salary
          from dw_employee
          group by dep_id
      )b
      where a.dep_id=b.dep_id and a.salary=b.max_salary
)
```

哪个部门的人数比销售部的人数多

```
select a.dep_id
      ,a.p_num
      ,c.sp_num
from (select dep_id
            ,count(1) p_num
    from dw_employee
    group by dep_id
    ) a
    ,(
    select dep_id
          ,count(1) sp_num
    from dw_employee b
    where b.dep_id in(
        select dep_id from dep where dep_name='销售部'
    )
    group by dep_id
    ) c
where a.p_num > c.sp_num
```

作业

- 哪些部门的平均薪水比销售部的平均薪水高
- 平均工资大于6000的部门
- 哪些人的薪水低于公司的平均水平

哪些人是其他人的上级

```
select *
from dw_employee a
where exists(
    select emp_id
    from dw_employee
    where leader_id = a.emp_id
)
```

哪些人不是其他人的上级

```
select *
from dw_employee a
where not exists(
    select 1
    from dw_employee
    where leader_id = a.emp_id
)
```

哪些部门 没有员工

```
select *
from dep a
where not exists (
    select 1
    from dw_employee
    where dep_id =a.dep_id
)
```

join和union的区别

join 是两张表做交连后里面条件相同的部分记录产生一个记录集，
union是产生的两个记录集(字段要一样的)并在一起，成为一个新的记录集 。
可以理解为：Union是纵向扩展，join上横向扩展
没有关联条件的join就是笛卡尔乘积

join

JOIN用于按照ON条件联接两个表，主要有四种：
INNER JOIN：内部联接两个表中的记录，仅当至少有一个同属于两表的行符合联接条件时，内联接才返回行。我理解的是只要记录不符合ON条件，就不会显示在结果集内。
LEFT JOIN / LEFT OUTER JOIN：外部联接两个表中的记录，并包含左表中的全部记录。如果左表的某记录在右表中没有匹配记录，则在相关联的结果集中右表的所有选择列表列均为空值。理解为即使不符合ON条件，左表中的记录也全部显示出来，且结果集中该类记录的右表字段为空值。
RIGHT JOIN / RIGHT OUTER JOIN：外部联接两个表中的记录，并包含右表中的全部记录。简单说就是和LEFT JOIN反过来。
FULL JOIN / FULL OUTER JOIN：完整外部联接返回左表和右表中的所有行。就是LEFT JOIN和RIGHT JOIN和合并，左右两表的数据都全部显示。

union

UNION运算符

> 将两个或更多查询的结果集组合为单个结果集，该结果集包含联合查询中的所有查询的全部行。UNION的结果集列名与UNION运算符中第一个Select语句的结果集的列名相同。另一个Select语句的结果集列名将被忽略。

其中两种不同的用法是**UNION**和**UNION ALL**，区别在于UNION从结果集中删除重复的行。如果使用UNION ALL 将包含所有行并且将不删除重复的行。

UNION和UNION ALL的区别：
union 检查重复
union all 不做检查
比如 `select 'a' union select 'a'` 输出就是一行 a
比如 `select 'a' union all select 'a'` 输出就是两行 a

合集

薪水大于8000或者小于2000或者等于5000的员工

```
select * from dw_employee where salary > 8000
union all
select* from dw_employee where salary < 2000
union all
select * from dw_employee where salary = 5000
```

提取平均薪水大于5000或者地址在北京的部门的dep_id

```
-- 提取平均薪水大于5000，或者地址在北京的部门的dep_id  union
-- 提取平均薪水大于5000，并且地址在北京的部门的dep_id  intersect
-- 提取平均薪水大于5000，并且地址不在北京的部门的dep_id  minus
select dep_id
from employee
group by dep_id
having avg(salary) > 5000
union
select dep_id
from dep
where dep_address like '%北京%'
```

表关联

列出员工的姓名和所在的部门的名称和地址

```
select a.emp_name
      ,b.dep_name
      ,b.dep_address
from dw_employee a
join dep b
on a.dep_id = b.dep_id
--on a.dep_id = cast(b.dep_id as int)
```

列出所有员工的姓名和他上司的姓名

```
select a.emp_name,b.emp_name
from dw_employee a
join dw_employee b
on a.emp_id = b.leader_id
```

> 内连接是两张表根据关联条件相互过滤，能够关联上的数据才会出现在结果集中

列出员工的姓名和他所在的部门，把没有部门的员工也列举出来

```
select e.emp_name
      ,d.*
from dw_employee e
left join dep d
on e.dep_id=cast(d.dep_id as int)
```

全外连接

full outer join 两张表相互补充
列举员工以及部门信息，没有部门的员工也列举出来，没有员工的部门也列举出来

```
select a.emp_name
      ,b.*
from dw_employee a
full outer join dep b
on a.dep_id = cast(b.dep_id as int)
```