1. **SELECT语法：**

SELECT [ALL | DISTINCT] select\_expr, select\_expr, ...

FROM table\_reference

[WHERE where\_condition]

[GROUP BY col\_list [HAVING condition]]

[ CLUSTER BY col\_list

| [DISTRIBUTE BY col\_list] [SORT BY| ORDER BY col\_list]

]

[LIMIT number]

Hive支持的基本类型:

tinyint smallint int bigint

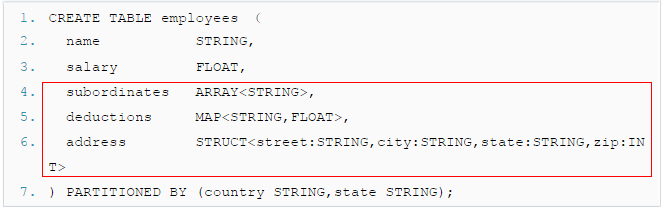
Boolean float double string

Hive支持的复杂数据类型:

1)array:一组有序字段.字段类型必须相同 Array(1,2)

2)map:一组无需的键/值对.键的类型必须是原子的,值可以是任何类型.同一个映射的键的类型必须相同,值的类型也必须相同 Map('a',1,'b',2)

3)struct:一组命名的字段,字段类型可以不同 Struct('a',1,1,0),创建包含复杂类型的表:



1. **设置reducer的个数**

设置reducer的个数为2,reducer的个数主要影响查询中的distribute by/cluster by/和sort by，具体影响在之后的介绍中做说明。设置reducerde的个数命令如下:

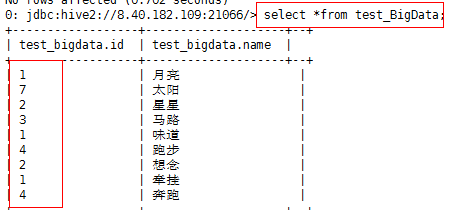
set hive.exec.reducers.bytes.per.reducer=2;

set hive.exec.reducers.max=2;

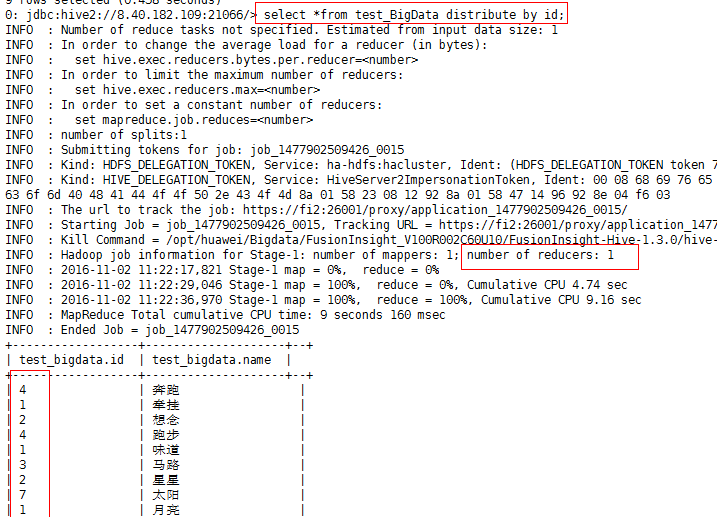
1. **distribute by**

distribute by指定的内容将数据分到同一个reducer

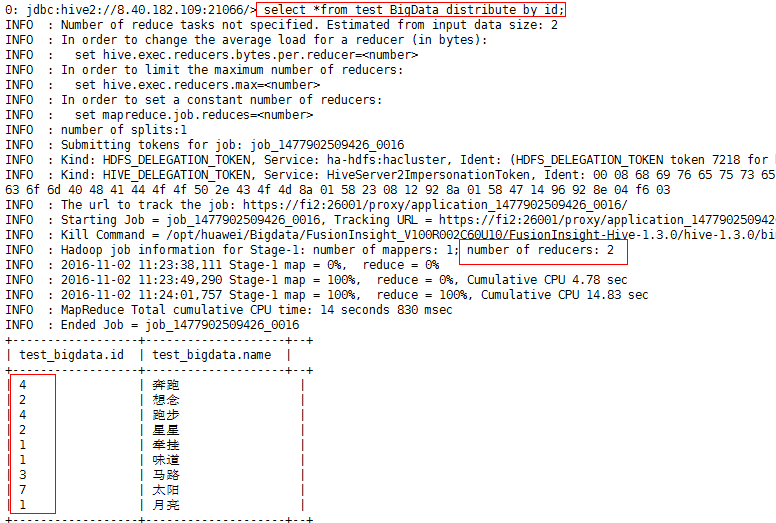
原始数据:



reducer为1:

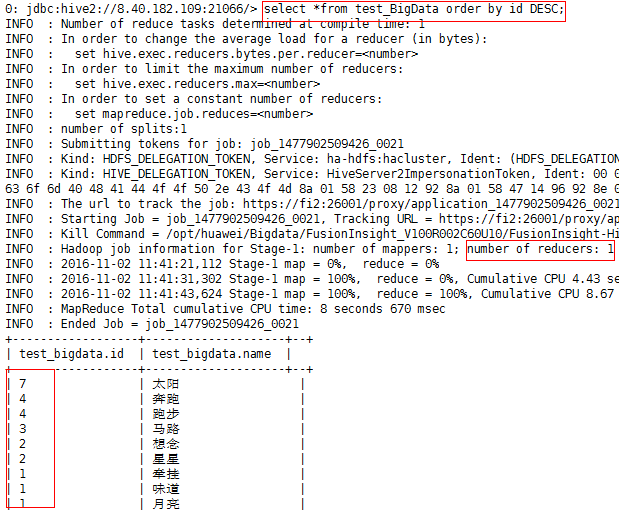


设置reducer为2:



1. **order by**

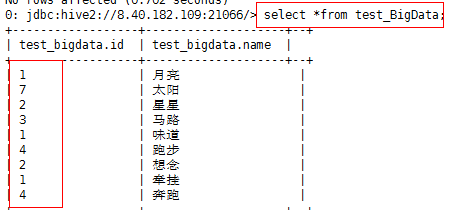
DESC降序排列,ASC升序排列，默认为ASC。Order by 会对输入做全局排序，因此只有一个reducer(多个reducer无法保证全局有序)。

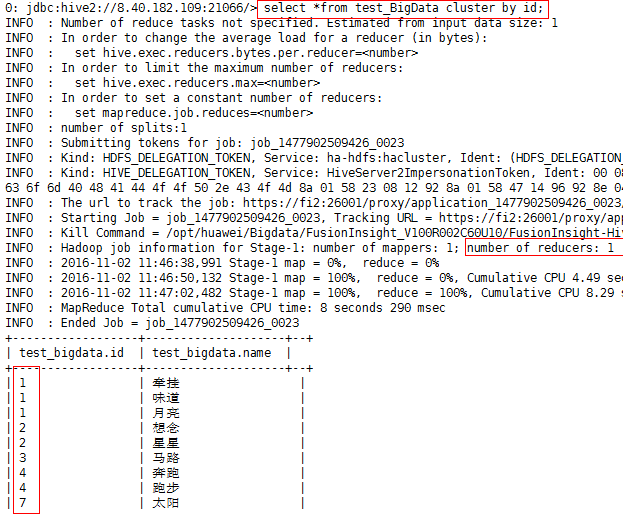


1. **cluster by**

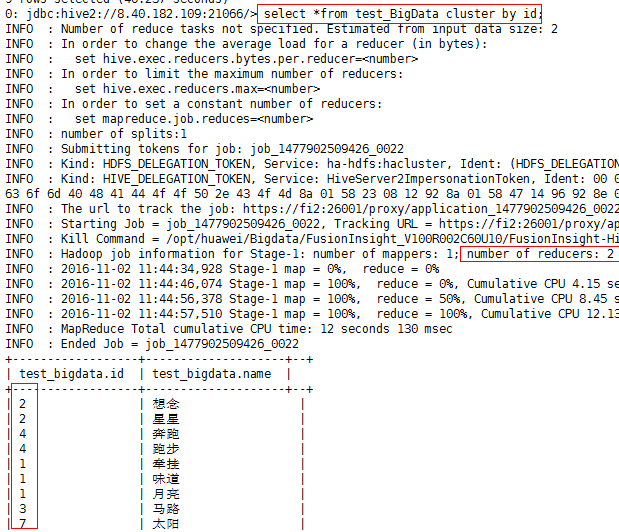
cluster by 除了具有distribute by功能外，还会该字段进行排序，因此常常认为cluster by=distribute by 和order by。

原始数据：



reduce为1，数据完全有序：

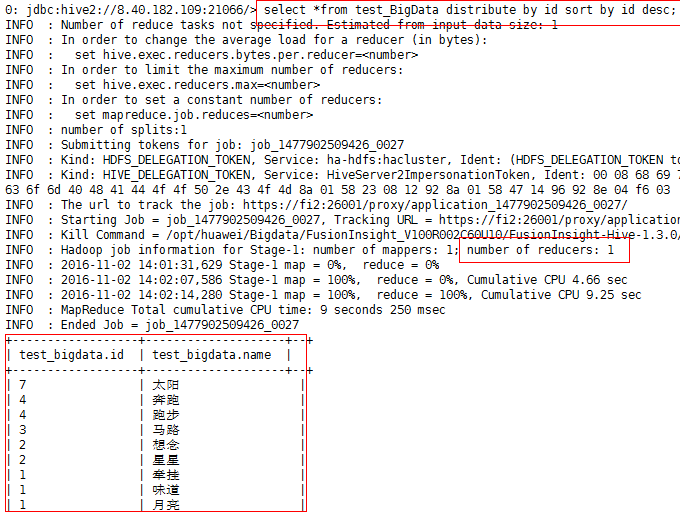
reduce为2,数据部分有序:



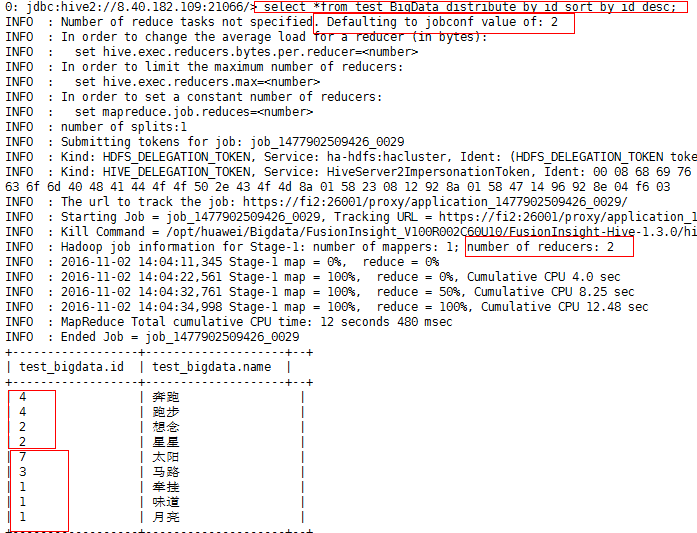
1. **sort by &.distribute by**

sort by不是全局排序,其在数据进入reducer之前完成排序。因此，如果用sort by进行排序，并且设置mapred.reduce.tasks>1,则sort by只保证每个reducer的输出有序，不保证全局有序

mapred.reduce.tasks=1时，全局有序：



mapred.reduce.tasks=2时，部分有序：



1. **group by**

注意:1.在sql命令格式使用的先后顺序上,group by先于order by.

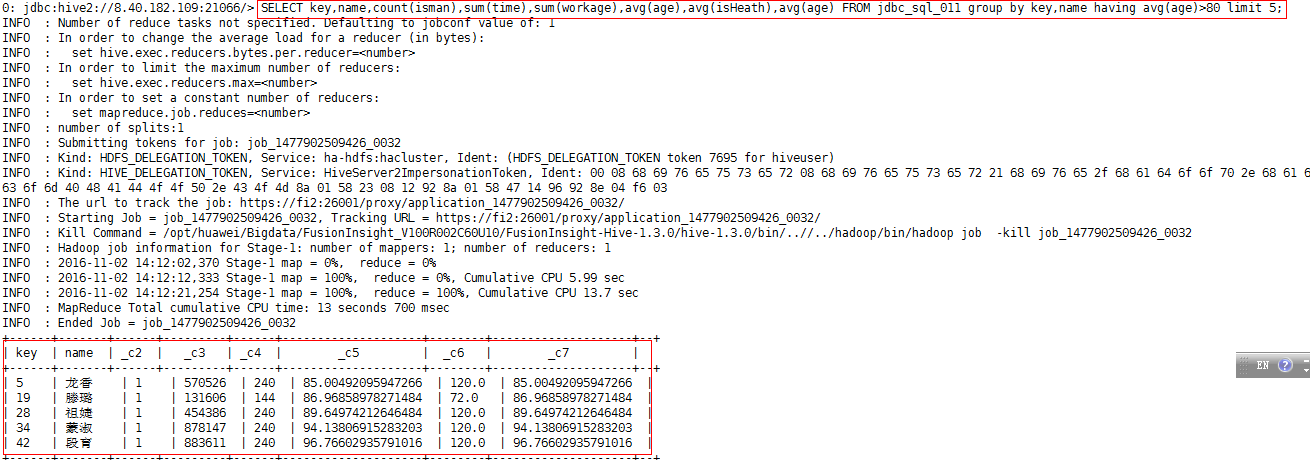
2.group by必须有”聚合函数”来配合才能使用,使用时至少需要一个分组标志字段.

3.聚合函数,例如:sum(),count(),avg();

4.使用group by的目的就是将数据分类汇总

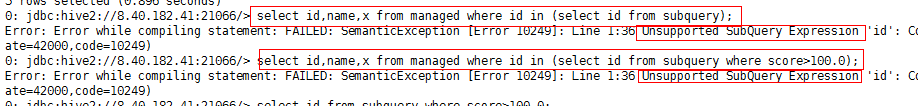
5.一般格式:select 单位名称,count(职工id),sum(职工工资) from 表名 group by 单位名称

6.group by有一个原则,就是select后面所有列中,没有使用聚合函数的列,必须出现在group by后面[重要]

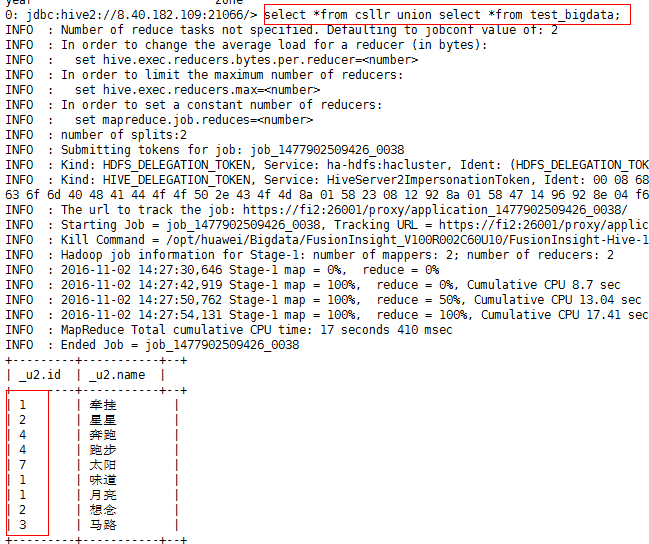


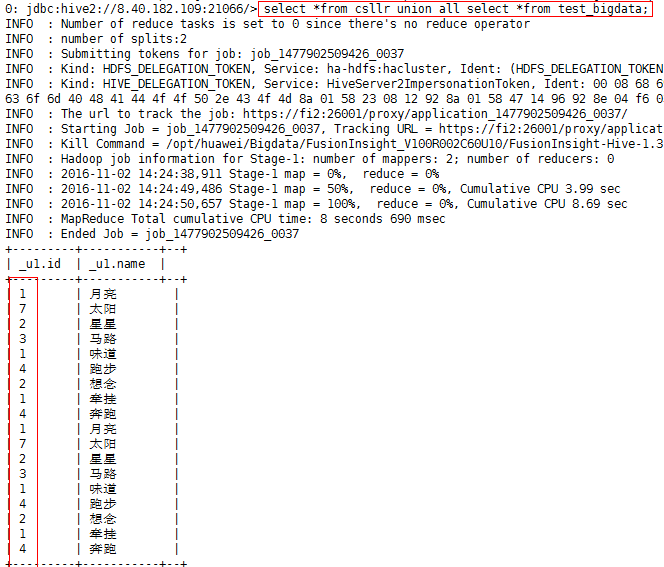
1. **subquery**

暂不支持子查询, 可以使用三个表的jion查询



1. **union**

union指令的目的是将两个sql语句的结果合并起来.union只是将两个结果联结起来一起显示,并不是联结两个表,类似select distinct。Union示例： union all 指令的目的也是将两个sql语句的结果合并起来. Union all 和union的不同之处在于,Union all 会将每一个符合条件的数据展示出来,无论数据是否重复。Union all示例：



1. **join**

只支持等值join:

SELECT a.\* FROM a JOIN b ON (a.id = b.id)

SELECT a.\* FROM a JOIN b ON (a.id = b.id AND a.department = b.department)；

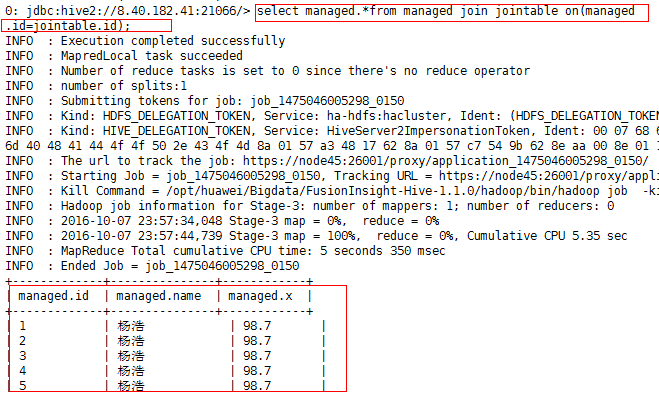
可以 join 多于 2 个表，如果join中多个表的 join key 是同一个，则 join 会被转化

为单个 map/reduce 任务.例如

SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key2) ;

分区join查询示例:

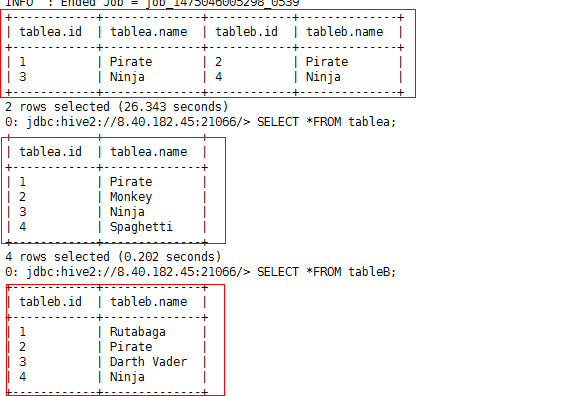
SELECT c.val, d.val FROM c LEFT OUTER JOIN d ON (c.key=d.key AND d.ds='2009-07-07' AND c.ds='2009-07-07') ; LEFT SEMI JOIN 的限制是， JOIN 子句中右边的表只能在 ON 子句中设置过滤条件，在 WHERE 子句、SELECT 子句或其他地方过滤都不行。



1. **inner join**

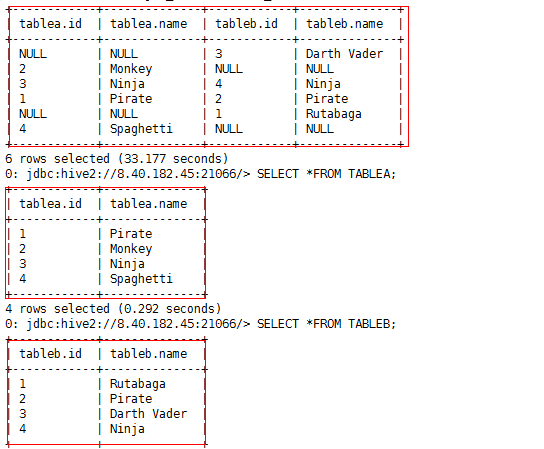
Inner join 产生的结果集中，是A和B的交集。

SELECT \* FROM TableA INNER JOIN TableB ON TableA.name = TableB.name;



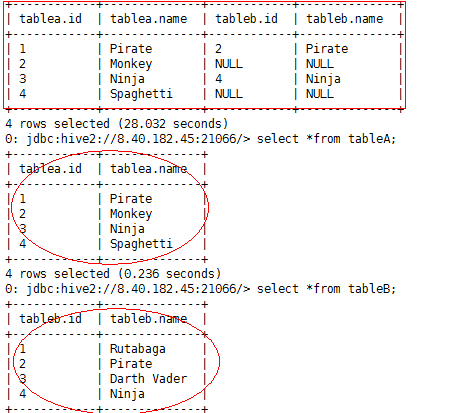
1. **Full outer join**

Full outer join 产生A和B的并集。但是需要注意的是，对于没有匹配的记录，则会以null做为值. 例如：SELECT \* FROM TableA FULL OUTER JOIN TableB ON TableA.name = TableB.name;

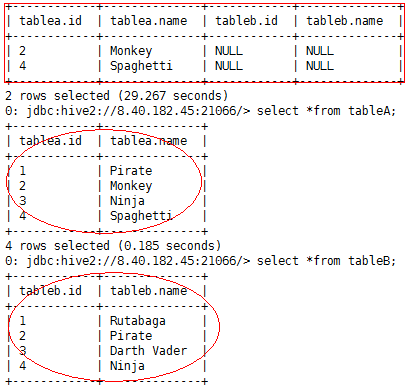


1. **outer join**

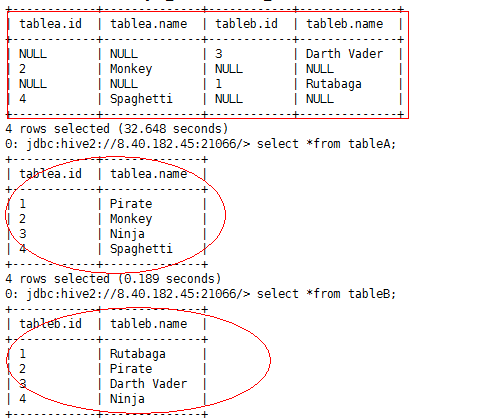
Left outer join 产生表A的完全集，而B表中匹配的则有值，没有匹配的则以null值取代。例如：SELECT \* FROM TableA LEFT OUTER JOIN TableB ON TableA.name = TableB.name;



SELECT \* FROM TableA LEFT OUTER JOIN TableB ON TableA.name = TableB.name WHERE TableB.id IS null；产生在A表中有而在B表中没有的集合。

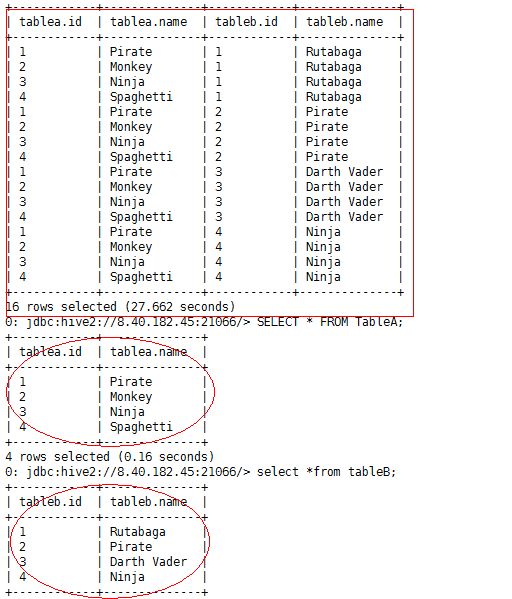


SELECT \* FROM TableA FULL OUTER JOIN TableB ON TableA.name = TableB.name WHERE TableA.id IS null OR TableB.id IS null;产生A表和B表都没有出现的数据集。



1. **cross join**

笛卡尔积： cross join,其就是把表A和表B的数据进行一个N\*M的组合，即笛卡尔积。一般来说，我们很少用到这个语法。但是我们得小心，如果不是使用嵌套的select语句，一般系统都会产生笛卡尔乘积然再做过滤。这是对于性能来说是非常危险的，尤其是表很大的时候。

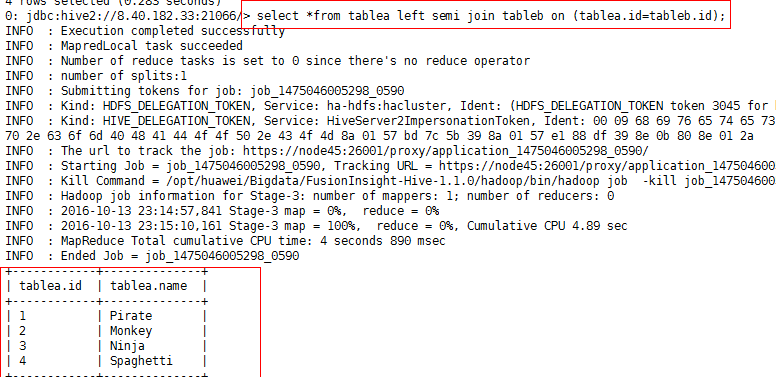


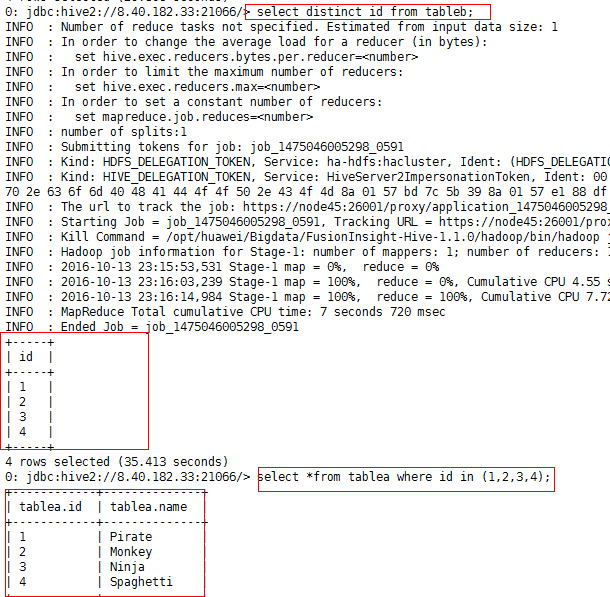
1. **left semi join**

**left semi join:** 是 IN/EXISTS 子查询的一种更高效的实现。Hive 当前没有实现 IN/EXISTS 子查询，所以你可以用 LEFT SEMI JOIN 重写你的子查询语句。LEFT SEMI JOIN 的限制是， JOIN 子句中右边的表只能在 ON 子句中设置过滤条件，在 WHERE 子句、SELECT 子句或其他地方过滤都不行。例如：

select \*from tablea left semi join tableb on (tablea.id=tableb.id)；等价于：

select \*from tablea where id in(select id from tableb);hive不支持此方式

****

****

Join学习网址：http://3ms.huawei.com/hi/group/2025905/thread\_5012843.html?mapId=5853695