# Hive7 查询

## SELECT ... FROM 语句

### 普通查询

- select 查询数组的时候

```
hive> SELETC name, suborinates FROM employees;
Jhon Doe        ["Mary Smith","Todd Jones"]
Mary Smith      ["Bill King"]
Todd Jones      []
Bill King       []
```

- select 查询map

```
hive> SELECT name, deductions FROM employees;
John Doe        {"Federal Taxes":0.2, "State Taxes":0.05, "Insurance":0.1}
Mary Simth      {"Federal Taxes":0.2, "State Taxes":0.05, "Insurance":0.1}
Todd Jones      {"Federal Taxes":0.15, "State Taxes":0.03, "Insurance":0.1}
```

- select 查询 struct

```
hive> SELECT name, adress FROM employees;
John Doe        {"street":"1MichiganAve.","city": "Chicago","state":"IL",
"zip":60600}
Mary Simth      {"street":"1MichiganAve.","city": "Chicago","state":"IL",
"zip":60600}
Todd Jones      {"street":"1MichiganAve.","city": "Chicago","state":"IL",
"zip":60600}
```

### 操作集合类型中的元素

- 基于数组索引进行查询

```
hive> SELECT name, subordinates[0] FROM employees;
John Doe        Mary Simth
Mary Simth      Bill King
Todd Jones      NULL
Bill King       NULL
```

- 基于map索引进行查询

```
hive> SELECT name, deductions["State Taxes"] FROM employees;
John Doe        0.05
Mary Simth      0.05
Todd Jones      0.03
Bill King       0.03
```

- 基于struct索引进行查询

```
hive> SELECT name, address.city FROM employees;
John Doe        Chicago
Mary Simth      Chicago
Todd Jones      Oak Park
Bill King       Obscuria
```

## 使用正则表达式来指定列

```
hive> SELECT symbol, `price.*` FROM stocks;
AAPL   195.69   197.88  194.0    194.12
AAPL   195.69   197.88  194.0    194.12
AAPL   195.69   197.88  194.0    194.12
```

## 使用列值进行计算

```
hive> SELECT upper(name), salary, deduction["Federal Taxes"],
    > round(salary * (1 - deductions["Federal Taxes"])) FROM employees;
JOHN DOE   10000.0   0.2  80000
JOHN DOE   10000.0   0.2  80000
JOHN DOE   10000.0   0.2  80000
```

## 使用函数

- 数学函数

```
hive> SELECT upper(name), salary, deduction["Federal Taxes"],
    > round(salary * (1 - deductions["Federal Taxes"])) FROM
```

| 返回值类型 | 样式 | 描述 |
| --- | --- | --- |
| BIGINT | round(DOUBLE d) | 返回DOUBLE型d的BIGINT类型的近似值 |
| … | … | … |

详细的话可以看 hive编程指南

- 聚合函数

对多行进行一些计算，然后得到一个结果值

```
hive> SELECT count(*), avg(salary) FROM employees;
4  77500.0
```

详细的话可以看 hive编程指南

- 表生成函数

可以将单列扩展多列或者多行

```
hive> SELECT explode(subordinates) AS sub FROM employees;
Mary Smith
Todd Jones
Bill King
```

## LIMIT语句

限制输出个数

```
SELECT count(*), avg(salary) FROM employees LIMIT 2;
```

## 列别名

给列取个名

```
SELECT count(*) as count_nums,
avg(salary) as avr_salary
FROM employees;
```

## 嵌套SELECT语句

```
hive> FROM(
    >   SELECT upper(name), salary, deductions["Federal Taxes"] as fed_taxes
    >   FROM employees
    >   )  e
    >   SELECT e.name, e.salary_minus_fed_taxes
    >   WHERE e.salary_minus_fed_taxes > 70000;

JOHN DOE 100000.0   0.2   80000
```

## CASE...WHEN..THEN..句式

```
hive> SELECT name, salary,
    >   CASE
    >   WHERE salary < 5000.0 THEN 'low'
    >   WHERE salary >= 5000.0 AND salary < 7000.0 THEN 'middle'
    >   ELSE 'very high'
    >   END AS bracket FROM employees;

John Doe            10000.0  very high
Mary Smith          6000.0   middle
```

## 什么情况下Hive可以避免进行MapReduce

- 简单读取employees对应的存储目录下的文件

```
SELECT * FROM employees;
```

- 过滤条件只是分区字段这种情况

```
SELECT * FROM employees
WHERE country = 'US' AND state = 'CA'
LIMIT 100;
```

其他都是由MapReduce来执行其他所有的查询。

# WHERE 语句

## 谓词操作符

| 操作符 | 支持的数据类型 | 描述 |
| --- | --- | --- |
| A = B | 基本数据类型 | A等于B返回1，否则0 |
| A<=>B | 基本数据类型 | A和B都为NULL则返回TRUE，任何一个为NULL 为NULL |
| … | … | … |

注意： A==B 这个是错误的

## 关于浮点数比较

```
hive> SELECT name, salary, deductions['Federal Taxes']
    > FROM employees WHERE deductions['Federal Taxes'] > 0.2;

John Doe        100000.0   0.2
Mary Smith      8000.0     0.2
Boss Man        20000.0    0.3
```

为什么 deductions['Federal Taxes'] = 0.2的记录被输出了?

实际上可以抽象来说，0.2对于FLOAT类型是0.20000001，而对于DOUBLE类型是0.200000000001。这是因为8个字节的DOUBLE值具有最多的小数位。当表中的FLOAT值通过Hive转换为DOUBLE值时，其产生的DOUBLE值是0.200000001000，要比0.2000000000001.这就是为什么比较打了。

解决办法:

- 在表中定义字段类型为DOUBLE而不是FLOAT
- 使用cast操作符

  ```
  hive> SELECT name, salary,deductions['Feberal Taxes'] FROM employees
      > WHERE deductions['Federal Taxes'] > cast(0.2, FLOAT);
  ```

- 和钱相关都不要使用浮点数

## LIKE 和 RLIKE

> LIKE 匹配操作符

```
hive> SELECT name, adress.street FROM employees WHERE adress.street LIKE '%Ave.'

John Doe    1 Michigan Ave
```

> RLIKE 可以通过Java的正则表达式这个更强大的语言来指导匹配条件

```
hive> SELECT name, adrees.street
    > FROM empoyees WHERE adress.street RLIKE '.*(Chicage|Ontario).*';

Mary Smith    100 Ontario  St.
```

## GROUP BY 语句

```
hive> SELECT year(ymd), avg(price_close) FROM stocks
    > WHERE exchange = 'NASDAQ' AND symbol = 'AAPL'
    > GROUP BY year(ymd);

1984     25.577776
1985     20.122921
```

## HAVING 语句

HAVING子句允许用户通过一个简单的语法完成原本需要通过子查询才能对GROUP BY 语句产生的分组进行条件过滤的任务。

```
hive> SELECT year(ymd), avg(price_close) FROM stocks
    > WHERE exchange = 'NASDAQ' AND symbol = 'AAPL'
    > GROUP BY year(ymd)
    > HAVING avg(price_close) > 50.0
```

如果没有HAVING语句：

```
hive> SELECT year(ymd), avg(price_close) FROM stocks
    > WHERE exchange = 'NASDAQ' AND symbol = 'AAPL'
    > GROUP BY year(ymd) s2
    > WHERE s2.avg > 50.0;
```

# JOIN 语句

## INNER JOIN

> 内连接（INNER JOIN）中，只有进行连接的两个表中都存在与连接标准相匹配的数据才会被保留下来。

```
对查询对苹果公司的股价（股票代码AAPL）和IBM公司的股价（股票代码IBM）进行比较。
hive> SELECT a.ymd, a.price_close, b.price_close
    > FROM stocks a JOIN stocks b ON a.ymd = b.ymd
    > WHERE a.symbol = 'APPL' AND b.symbol = 'IBM';
2010-01-04    231.12  1231.21
2020-02-02    231.12  1231.21
```

- ON子句制定了两个表间数据进行连接的条件,目前还不允许使用OR
- WHERE子句限制了左边表是AAPL的记录，右边表是IBM的记录

```
hive> SELECT s.ymd, s.symbol, s.price_close, d.dividend
    > FROM stocks s JOIN dividends d ON s.ymd=d.ymd AND s.symbol = d.symbol
    > WHERE s.symbol = 'APPL';

1987-05-11  AAPL  77.0   0.015
1987-05-11  AAPL  77.0   0.015
1987-05-11  AAPL  77.0   0.015
```

```
hive> SELECT a.ymd, a.price_close, b.price_close, c.price_close
    > FROM stocks a JOIN stocks b ON a.ymd = b.ymd
                    JOIN stocks c ON a.ymd = c.ymd
    > WHERE a.symbol = 'APPL' AND b.symbol = 'IBM' AND c.symbol = 'GE';
```

大多数情况下，Hive会对每对JOIN连接对象启动一个MapReduce任务。在上面的例子中，首先先启动一个MapReduce job 对表a 和 表 b 进行连接操作。再启动一个MapReduce Job将第一个表的输出和表c进行连接操作。

## JOIN 优化

> 在前面例子中，每个ON子句都是用到了a.ymd作为其中JOIN连接键。HIve可以通过一个优化可以在同一个MapReduce Job中连接三张表。
>
> 提示：当对3个或者更多个表进行JOIN连接时，如果每个ON子句都是用相同的连接键的话，只会产生一个MapReduce job

Hive 自己会假定查询中最后一个表是最大的表，在对每行记录进行连接操作的时候，它会尝试将其他表缓存起来，然后扫描最后那个表进行计算。用户也需要保证联系查询中的表大小是从左到右增加的。

- 第一种方法：

```
hive> SELECT s.ymd, s.symbol, s.price_close, d.dividend
    > FROM stocks s JOIN dividends d ON s.ymd=d.ymd AND s.symbol = d.symbol
    > WHERE s.symbol = 'APPL';

这里默认 dividends表比stock表更大
```

- 第二种方法

```
hive>SELECT /*+STREAMTABLE(S)*/ s.ymd, s.symbol, s.price_close, d.dividend
    > FROM stocks s JOIN dividends d ON s.ymd=d.ymd AND s.symbol = d.symbol
    > WHERE s.symbol = 'APPL';
```

## LEFT OUTER JOIN

左外连接通过关键字LEFT OUTER 进行标识：

```
hive> SELECT s.ymd, s.symbol, s.price_close, d.dividend
    > FROM stocks s LEFT OUTER JOIN dividents d ON s.ymd = d.ymd AND s.symbol =
d.symbol
    > WHERE s.symbol = 'AAPL';
...
1987-05-01  AAPL    80.0   NULL

1987-05-01  AAPL    80.0   0.0015
...
```

JOIN操作符左边表中符合WHERE子句的所有记录都会被返回。JOIN操作符右边表中如果没有符合ON后面连接条件的记录时，那么从右边表指定选择的列的值会是NULL。

## OUTER JOIN

对于外连接（OUTER JOIN）会忽略掉分区过滤条件。不过，对于内连接（INNER JOIN）使用过滤谓词确实是起作用的。

使用嵌套进行SELECT语句：

```
hive> SELECT s.ymd, s.symbol, s.price_close, d.dividend FROM
    > (SELECT * FROM stocks WHERE symbol = 'AAPL' AND exchange = 'NASDAQ') s
    > LEFT OUTER JOIN
    > (SELECT * FROM dividends WHERE symbol = 'AAPL' AND exchange = 'NASDAQ') d
    > ON s.ymd = d.ymd;

...
1988-02-10   AAPL   41.0   NULL
...
```

> WHERE语句在连接操作执行之后会执行，因此WHERE语句应该只用于过滤那些非NULL值得列值。

### RIGHT OUTER JOIN

右外连接会返回右边表所有符合WHERE语句的记录。坐标中匹配不上的字段用NULL代替。

```
hive> SELECT s.ymd, s.symbol, s.price_close, d.dividend
    > FROM dividents d RIGHT OUTER JOIN stocks s ON s.ymd = d.ymd AND s.symbol =
d.symbol
    > WHERE s.symbol = 'AAPL';
...
1987-05-01  AAPL    80.0   NULL

1987-05-01  AAPL    80.0   0.0015
...
```

### FULL OUTER JOIN

完全外连接（FULL OUTER JOIN）会返回所有表中符合WHERE语句条件的所有记录。如果任意一个表没有符合条件的值的话，用NULL代替。

```
hive> SELECT s.ymd, s.symbol, s.price_close, d.dividend
    > FROM dividents d FULL OUTER JOIN stocks s ON s.ymd = d.ymd AND s.symbol =
d.symbol
    > WHERE s.symbol = 'AAPL';
...
1987-05-01  AAPL    80.0   NULL
1987-05-01  AAPL    80.0   0.0015
...
```

## LEFT SEMI-JOIN

左半开连接会返回左边表的记录，前提是记录对于右边表满足ON语句中的判定条件。

在HIVE中不支持的查询：

```
SELECT s.ymd, s.symbol, s.price_close FROM stocks s
WHERE s.ymd, s.symbol IN
(SELECT d.ymd, d.symbol FROM dividends d);
```

但是可以这么用：

```
hive> SELECT s.ymd, s.symbol, s.price_close
    > FROM stocks s LEFT SEMI JOIN dividents d ON s.ymd = d.ymd AND s.symbol =
d.symbol;
```

注意，SELECT 和 WHERE语句不能引用到右边表中的字段。

> SEMI-JOIN 比通常的INNER JOIN更高效，原因如下: 对于左表中一条指定的记录，在右表中一旦找到匹配的记录，Hive就会立刻停止扫描。

## 笛卡尔积 JOIN

笛卡尔积是左表的行数乘以右边表的行数等于笛卡尔积数据集的大小。会执行很长时间。

```
hive> SELECT * FROM stocks JOIN dividends
    > WHERE stock.symbol = dividends.symbol and stock.symbol = 'APPL';
```

> 应用场景： 假设一个表表示用户偏好，另一个表表示新闻文章，同时有一个算法会推测出用户可能会喜欢读哪些文章。

## map-side JOIN

> 背景：如果所有表中只有一张表是小表，那么可以在最大的表通过mapper的时候将小表完全放到内存中。

> 原因：Hive可以和内存中的小表进行逐一匹配，从而省略掉常规连接操作所需要的reduce过程。

```
hive> set hive.auto.convert.join=true;

hive>SELECT /*+ MAPJOIN(d)*/ s.ymd, s.symbol, s.price_close, d.dividend
    >FROM stocks s JOIN dividends d ON s.ymd = d.ymd AND s.symbol = d.symbol
    >WHERE s.symbol = 'AAPL';
```

这个执行速度快了大概30%;

# ORDER BY 和 SORT BY

- ORDER BY

> ORDER BY 语句和其他的SQL方言中的定义是一样的，其会对查询结果执行一个全局排序，也就是说会有一个所有的数据都通过一个reducer进行处理的过程。大的数据集，会执行很长时间。

```
SELECT s.ymd, s.symbol, s.price_close
FROM stocks s
ORDER BY s.ymd ASC, s.symbol DESC;
```

- SORT BY

> 只会在每个reducer中队数据进行排序，也就是执行一个局部排序的过程，这可以保证每个reducer的输出数据都是有序的。可以提高后面进行的全局排序的效率。

```
SELECT s.ymd, s.symbol, s.price_close
FROM stocks s
SORT BY s.ymd ASC, s.symbol DESC;
```

# 含有SORT BY 的 DISTRIBUTE BY

> 背景： 假设我们希望具有相同股票交易码的数据在一起处理，那么我们可以使用DISTRBUTE BY来保证具有相同的股票交易码的记录会分发到同一个reducer中进行处理，再使用SORT BY来按照我们的期望进行排序

```
hive> SELECT s.ymd, s.symbol, s.price_close
    > FROM stocks s
    > DISTRIBUTE BY s.symbol
    > SORT BY s.symbol ASC, s.ymd ASC;
1984-09-07  AAPL  26.5
1984-09-10  AAPL  26.37
1984-09-11  AAPL  26.87
```

# CLUSTER BY

> 前面的例子中，s.symbol列被用在了DISTRIBUTE BY语句中，而s.symbol 列和s.ymd位于SORT BY语句中，如果这两个语句涉及到列完全相同，而且采用的是升序排序方式。那么CLUSTER BY就是等价于前面两个语句

```
hive> SELECT s.ymd, s.symbol, s.price_close
    > FROM stocks s
    > CLUSTER BY s.symbol
2010-02-08 AAPL 194.12
2010-02-05 AAPL 195.12
```

# 类型转换

## CAST 类型转换

- 字符串转化为FLOAT类型

```
SELECT name, salary FROM employees
WHERE cast(salary AS FLOAT) < 100000.0
```

如果里面的值是不合法的，那么Hive会返回null。

浮点数转化为整数：一般用round()或者floor()；

## BINARY 值

- BINARY类中转化为STRING类型

```
SELECT(2.0*cast(cast(b as string)as double)) from src;
```

# 抽样查询

## 分桶抽样

> 背景：对于非常大的数据集，HIve可以通过对表进行分桶抽样来满足这个需求。

- 使用rand() 函数进行抽样

  ```
  SELECT * FROM numbers TABLESAMPLE(BUCKET 3 OUT OF 10 ON rand()) s;
  ```

## 数据块抽样

- 基于行数进行抽样

  ```
  hive> SELECT * FROM numbersflat TABLESAMPLE(0.1 PERCENT) s;
  ```

  这种抽样的最小抽样单元是一个HDFS数据块，如果数据块的大小小于128M的话，会返回所有行。

# UNION ALL

> UNION ALL 可以将2个或者多个表进行合并，每个union子查询都需具有相同的列，而且对应的类型必须是一致的

数据进行合并

```
SELECT log.ymd, log.level, log.message
        FROM(
        SELECT l1.ymd, l1.level,
            l1.message, 'Log1' AS source
        FROM log1 l1
    UNOIN ALL
        SELECT l2.ymd, l2.level,
            l2.message, 'Log2' AS source
        FROM log1 l2
    )log
SORT BY log.ymd ASC;
```