<课前话题>：在互联网公司当分析师是怎样一种感受？（15min）
- 分析师的种类
- 分析师的具体工作
- 分析师需要的技能

第一节——HiveSQL基础技能1
- 课堂目标
- HIVE简介(10min)
  - 1、HIVE是基于hadoop的数据仓库
  - 2、MapReduce简介
- 基础语法(15min)
  - 1、SELECT …A… FROM …B… WHERE …C…
  - 2、GROUP BY
  - 3、ORDER BY
  - 4、执行顺序
- 常用函数(35min)
  - 1、如何把时间戳转化为日期?
  - 2、如何计算日期间隔?
  - 3、条件函数
  - 4、字符串函数
  - 5、聚合统计函数
- 重点练习(15min)
- 常见错误及处理办法(10min)
  - 1、标点符号错误
  - 2、没有对子查询的表进行重命名
  - 3、使用错误的字段名
  - 4、丢了逗号分隔符
- 总结
- 作业

<课前话题>：在互联网公司当分析师是怎样一种感受？（15min）

分析师的种类

- 负责支持产品、运营团队的分析师(角色：business partner)
- 负责支持集团管理层的分析师
- 负责行业研究的分析师

分析师的具体工作

支持产品团队：

- 产品迭代的效果分析
- ABtest的实验设计与结果评估
- 产品改版方案的策略输出

支持运营团队：

- 活动运营的效果分析
- 用户运营的目标用户筛选
- 拉新、复购的策略输出

支持集团管理层：

- 公司整体的宏观情况分析
- 不同业务的横纵向对比分析
- 战略方向的策略输出

负责行业研究：

- 某个行业的大盘情况
- 竞对的供给、策略、产品等分析
- 我们是否有新的机会和增长点

分析师需要的技能

- SQL，数据的提取、清洗、加工
- 数据可视化
- 分析报告撰写
- 对业务的理解与思考

第一节——HiveSQL基础技能1

课堂目标

1. 掌握HIVE基础语法
2. 掌握HIVE常用函数
3. 掌握HIVE常用函数的组合使用
4. 掌握常见错误

HIVE简介(10min)

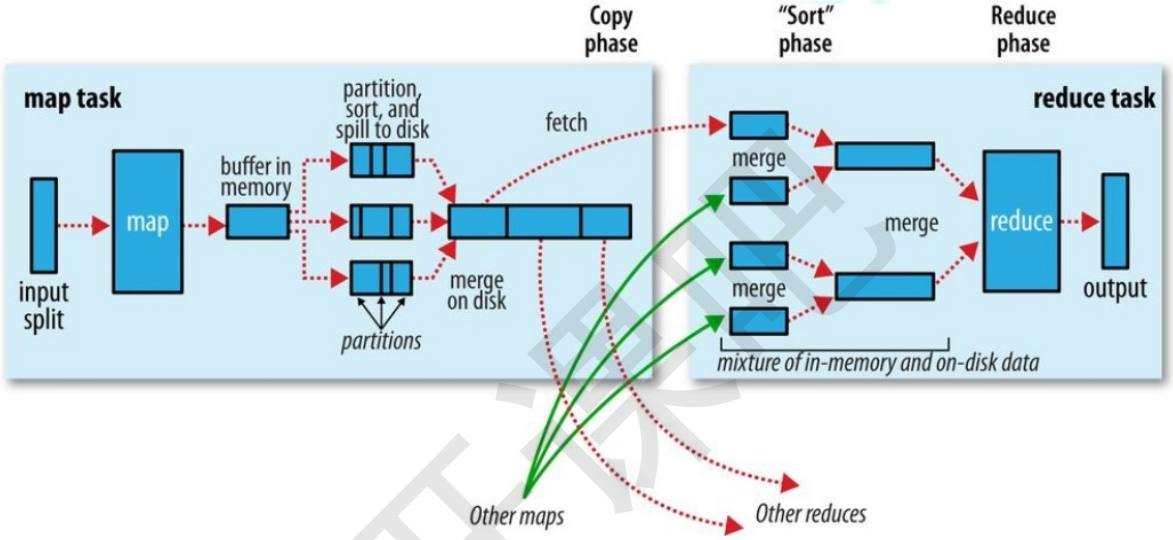
1、HIVE是基于hadoop的数据仓库

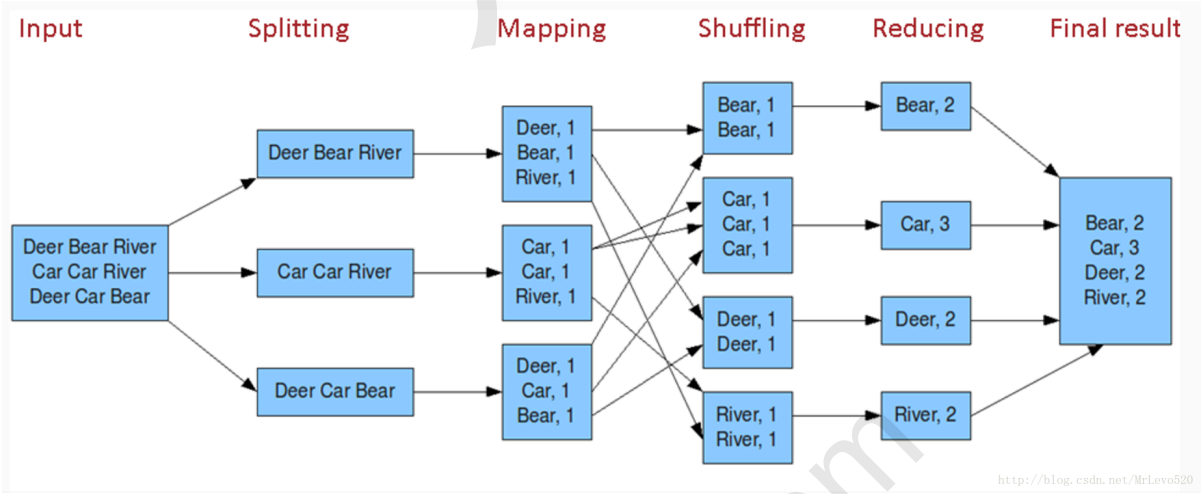
HiveSQL与传统SQL的对比：

查询语言	HQL	SQL
数据存储位置	HDFS	块设备或者本地文件
数据格式	用户定义	系统决定(不同的数据库有不同的存储引擎)
数据更新	不支持	支持
索引	无	有
执行	MapReduce	Executor
执行延迟	高	低
可扩展性	高	低
数据规模	大	小

PS：块设备是i/o设备中的一类，是将信息存储在固定大小的块中，每个块都有自己的地址，还可以在设备的任意位置读取一定长度的数据，例如硬盘,U盘，SD卡等。

2、MapReduce简介





基础语法(15min)

1、SELECT ...A... FROM ...B... WHERE ...C...

- **A**：列名
- **B**：表名
- **C**：筛选条件

user_info 列名	举例
user_id	10001,10002(唯一的)
user_name	Amy, Dennis(唯一的)
sex	[male, female]
age	[13,70]
city	beijing, shanghai
firstactivetime	2019-04-19 15:40:00
level	[1,10]
extra1	string类型：{"systemtype":"ios","education":"master","marriage_status":"1","phonebrand":"iphone X"}
extra2	map<string,string>类型：{"systemtype":"ios","education":"master","marriage_status":"1","phonebrand":"iphone X"}

```
hive> desc user_info;
OK
user_id                 string
user_name               string
sex                     string
age                     int
city                    string
firstactivetime         string
level                   int
extra1                  string
extra2                  map<string,string>
Time taken: 0.156 seconds, Fetched: 9 row(s)
```

```sql
--选出城市在北京，性别为女的10个用户名--
SELECT user_name
FROM user_info
WHERE city='beijing' and sex='female'
limit 10;
```

```
hive> SELECT user_name
    > FROM user_info
    > WHERE city='beijing' and sex='female'
    > limit 10;
OK
Angela
Becky
Betty
Cheryl
Christine
Corrine
DARCY
Janet
JUNE
Lena
Time taken: 1.972 seconds, Fetched: 10 row(s)
```

user_trade列名	举例
user_name	Amy, Dennis
piece	购买数量
price	价格
pay_amount	支付金额
goods_category	food, clothes, book, computer, electronics, shoes
pay_time	1323308943，时间戳
dt	partition，‘yyyy-mm-dd’

```
hive> desc user_trade;
OK
user_name               string
piece                   int
price                   double
pay_amount              double
goods_category          string
pay_time                bigint
dt                      string

# Partition Information
# col_name              data_type               comment
dt                      string
Time taken: 0.067 seconds, Fetched: 11 row(s)
```

注意：如果该表是一个分区表，则**WHERE**条件中必须对分区字段进行限制。

```
--选出在2019年4月9日，购买的商品品类是food的用户名、购买数量、支付金额--
SELECT user_name,
       piece,
       pay_amount
FROM user_trade
WHERE dt='2019-04-09' and goods_category='food' ;
```

未对分区进行限制的报错：

```sql
SELECT user_name,
       piece,
       pay_amount
FROM user_trade
WHERE goods_category='food' ;
```

```
No partition predicate for Alias "user_trade" Table "user_trade"
```

注意！分区表必须限制分区字段！

2、GROUP BY

```sql
--2019年一月到四月，每个品类有多少人购买，累计金额是多少--
SELECT goods_category,
       count(distinct user_name) as user_num,
       sum(pay_amount) as total_amount
FROM user_trade
WHERE dt between '2019-01-01' and '2019-04-30'
GROUP BY goods_category;
```

GROUP BY 的作用：分类汇总

常用聚合函数：

1. count()：计数 count(distinct .....) 去重计数
2. sum()：求和
3. avg()：平均值
4. max()：最大值
5. min()：最小值

- **GROUP BY ..... HAVING**

```sql
--2019年4月，支付金额超过5万元的用户--
SELECT user_name,
       sum(pay_amount) as total_amount
FROM user_trade
WHERE dt between '2019-04-01' and '2019-04-30'
GROUP BY user_name HAVING sum(pay_amount)>50000;
```

HAVING：对GROUP BY的对象进行筛选

仅返回符合HAVING条件的结果

```
Carroll 58596.0
Cheryl  123144.0
Iris    188870.0
JUNE    519948.0
Mitchell        193314.0
Morris  166984.0
Scott   95546.0
Time taken: 26.922 seconds, Fetched: 7 row(s)
```

3、ORDER BY

```sql
--2019年4月，支付金额最多的TOP5用户--
SELECT user_name,
       sum(pay_amount) as total_amount
FROM user_trade
WHERE dt between '2019-04-01' and '2019-04-30'
GROUP BY user_name
ORDER BY total_amount DESC  limit 5;
```

```
JUNE    519948.0
Mitchell        193314.0
Iris    188870.0
Morris  166984.0
Cheryl  123144.0
Time taken: 47.613 seconds, Fetched: 5 row(s)
```

ASC：升序(默认)

DESC：降序

对多个字段进行排序：ORDER BY A ASC , B DESC

为什么ORDER BY 后面不直接写sum(pay_amount)而是用total_amount?

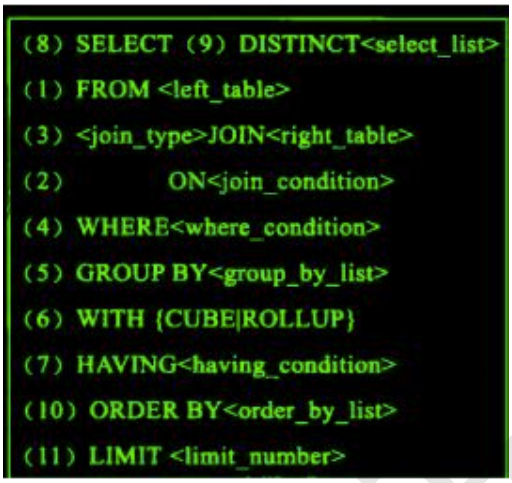
```sql
SELECT user_name,
       sum(pay_amount) as total_amount
FROM user_trade
WHERE dt between '2019-04-01' and '2019-04-30'
GROUP BY user_name
ORDER BY sum(pay_amount) DESC  limit 5;   ##错误写法##
```

不可以写：ORDER BY **sum(pay_amount)** DESC

——原因：**执行顺序！！！** ORDER BY的执行顺序在SELECT之后，所以需使用重新定义的列名进行排序。

4、执行顺序

FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY



常用函数(35min)

1、如何把时间戳转化为日期?

```
SELECT pay_time,
       from_unixtime(pay_time,'yyyy-MM-dd hh:mm:ss')
FROM user_trade
WHERE dt='2019-04-09';
```

```
hive> SELECT from_unixtime(pay_time,'yyyy-MM-dd hh:mm:ss')
    > FROM user_trade
    > WHERE dt='2019-04-09';
OK
2019-04-09 12:40:47
2019-04-09 06:59:58
2019-04-09 07:27:44
2019-04-09 03:36:06
2019-04-09 12:10:14
Time taken: 0.263 seconds, Fetched: 5 row(s)
```

- **from_unixtime(bigint unixtime, string format)**

format:

1. yyyy-MM-dd hh:mm:ss
2. yyyy-MM-dd hh
3. yyyy-MM-dd hh:mm
4. yyyyMMdd

拓展：把日期转化为时间戳——unix_timestamp

课后练习：unix_timestamp(string date)

2、如何计算日期间隔?

```
--用户的首次激活时间，与2019年5月1日的日期间隔--
SELECT user_name,
       datediff('2019-05-01',to_date(firstactivetime))
FROM user_info
limit 10;
```

```
hive> SELECT user_name,
    >        datediff('2019-05-01',to_date(firstactivetime))
    > FROM user_info
    > limit 10;
OK
Abby    383
Ailsa   332
Alice   148
Alina   45
Allison 440
Angelia 2
Amanda  440
Anne    523
Ann     498
Amy     420
Time taken: 0.347 seconds, Fetched: 10 row(s)
```

- **datediff(string enddate, string startdate)**：结束日期减去开始日期的天数

拓展：日期增加函数、减少函数——date_add、date_sub

- date_add(string startdate, int days)
- date_sub (string startdate, int days)

3、条件函数

- **case when**

统计以下四个年龄段**20**岁以下、**20-30**岁、**30-40**岁、**40**岁以上的用户数：

```
--统计以下四个年龄段20岁以下、20-30岁、30-40岁、40岁以上的用户数--
SELECT case when age<20 then '20岁以下'
            when age>=20 and age<30 then '20-30岁'
            when age>=30 and age<40 then '30-40岁'
            else '40岁以上' end,
       count(distinct user_id) user_num
FROM user_info
GROUP BY case when age<20 then '20岁以下'
            when age>=20 and age<30 then '20-30岁'
            when age>=30 and age<40 then '30-40岁'
            else '40岁以上' end;
```

```
20-30岁  48
20岁以下        37
30-40岁  62
40岁以上        180
Time taken: 24.896 seconds, Fetched: 4 row(s)
```

- **if**

统计每个性别用户等级高低的分布情况：

```
--统计每个性别用户等级高低的分布情况(level大于5为高级)--
SELECT sex,
       if(level>5,'高','低'),
       count(distinct user_id) user_num
FROM user_info
GROUP BY sex,
         if(level>5,'高','低');
```

```
female  低      80
female  高      97
male    低      83
male    高      67
Time taken: 23.884 seconds, Fetched: 4 row(s)
```

4、字符串函数

每个月新激活的用户数：

```
--每个月新激活的用户数--
SELECT substr(firstactivetime,1,7) as month,
       count(distinct user_id) user_num
FROM user_info
GROUP BY substr(firstactivetime,1,7);
```

```
2017-01 7
2017-02 14
2017-03 11
2017-04 15
2017-05 14
2017-06 10
2017-07 12
2017-08 20
2017-09 12
2017-10 11
2017-11 17
2017-12 7
2018-01 12
2018-02 5
2018-03 15
2018-04 10
2018-05 13
2018-06 14
2018-07 15
2018-08 11
2018-09 9
2018-10 9
2018-11 10
2018-12 12
2019-01 7
2019-02 12
2019-03 14
2019-04 8
2019-05 1
Time taken: 22.975 seconds, Fetched: 29 row(s)
```

- **substr(string A, int start, int len)**

备注：如果不指定截取长度，则从起始位一直截取到最后。

不同手机品牌的用户数：

extra1(string)：
{"systemtype":"ios","education":"master","marriage_status":"1","phonebrand":"iphone X"}

extra2(map<string,string>)：
{"systemtype":"ios","education":"master","marriage_status":"1","phonebrand":"iphone X"}

```sql
--不同手机品牌的用户数--
##第一种情况
SELECT get_json_object(extra1, '$.phonebrand') as phone_brand,
       count(distinct user_id) user_num
FROM user_info
GROUP BY get_json_object(extra1, '$.phonebrand');

##第二种情况
SELECT extra2['phonebrand'] as phone_brand,
       count(distinct user_id) user_num
FROM user_info
GROUP BY extra2['phonebrand'];
```

```
hive> desc user_info;
OK
user_id                 string
user_name               string
sex                     string
age                     int
city                    string
firstactivetime         string
level                   int
extra1                  string
extra2                  map<string,string>
Time taken: 0.156 seconds, Fetched: 9 row(s)
```

```
CHUIZI  35
HUAWEI  28
MI      35
VIVO    36
YIJIA   33
iphone4 20
iphone5 24
iphone6 27
iphone6s        24
iphone7 22
iphoneX 19
iphoneXS        24
Time taken: 25.002 seconds, Fetched: 12 row(s)
```

- **get_json_object(string json_string, string path)**

param1：需要解析的json字段

param2：用.key取出想要获取的value

5、聚合统计函数

ELLA用户的2018年的平均支付金额，以及2018年最大的支付日期与最小的支付日期的间隔：

```sql
--ELLA用户的2018年的平均支付金额，以及2018年最大的支付日期与最小的支付日期的间隔--
SELECT avg(pay_amount) as avg_amount,
       datediff(max(from_unixtime(pay_time,'yyyy-MM-dd')),min(from_unixtime(pay_time,'yyyy-MM-dd')))
FROM user_trade
WHERE year(dt)='2018'
      and user_name='ELLA';
```

max(from_unixtime(pay_time,'yyyy-MM-dd'))= from_unixtime(max(pay_time),'yyyy-MM-dd'))

注意：不许嵌套组合**avg(count(*))**

重点练习(15min)

1、2018年购买的商品品类在两个以上的用户数

```sql
SELECT count(a.user_name)
FROM
    (SELECT user_name,
           count(distinct goods_category) as category_num
    FROM user_trade
    WHERE year(dt)='2018'
    GROUP BY user_name HAVING count(distinct goods_category)>2)a;
```

2、用户激活时间在2018年，年龄段在20-30岁和30-40岁的婚姻状况分布

```sql
SELECT a.age_type,
       if(a.marriage_status=1,'已婚','未婚'),
       count(distinct a.user_id)
FROM

    (SELECT case when age<20 then '20岁以下'
                 when age>=20 and age<30 then '20-30岁'
                 when age>=30 and age<40 then '30-40岁'
                 else '40岁以上' end  as age_type,
           get_json_object(extra1, '$.marriage_status') as
marriage_status,
           user_id
    FROM user_info
    WHERE to_date(firstactivetime) between '2018-01-01' and '2018-12-31')a

WHERE a.age_type in ('20-30岁','30-40岁')
GROUP BY a.age_type,
         if(a.marriage_status=1,'已婚','未婚');
```

```
20-30岁 已婚    9
20-30岁 未婚    9
30-40岁 已婚    10
30-40岁 未婚    14
Time taken: 24.875 seconds, Fetched: 4 row(s)
```

常见错误及处理办法(10min)

1、标点符号错误

使用全角符号:

```
hive> SELECT *
    > FROM user_trade
    > WHERE dt='2019-04-09';
    >
hive> SELECT *
    > FROM user_trade
    > WHERE dt=‘2019-04-09’;
FAILED: ParseException line 3:9 character '‘' not supported here
line 3:20 character '’' not supported here
```

2、没有对子查询的表进行重命名

错误实例：

```
SELECT count(user_name)
FROM
    (SELECT user_name,
           count(distinct goods_category) as category_num
    FROM user_trade
    WHERE year(date)='2019'
    GROUP BY user_name HAVING count(distinct goods_category)>2);
```

3、使用错误的字段名

```
hive> SELECT username
    > FROM user_info
    > LIMIT 10;
FAILED: SemanticException [Error 10004]: Line 1:7 Invalid table alias or column reference 'user
name': (possible column names are: user_id, user_name, sex, age, city, firstactivetime, level,
extra1, extra2)
```

4、丢了逗号分隔符

```
hive> select user_name
    >        sex
    >        level
    > from user_info limit 10;
FAILED: ParseException line 3:7 missing EOF at 'level' near 'sex'
```

总结

1. 利用GROUP BY做聚合计算
2. 利用ORDER BY做排序
3. 牢记SQL执行顺序
4. 常用函数组合使用
5. 避免常见错误

作业

作业1：激活天数距今超过300天的男女分布情况(使用user_info)

作业2：不同性别、教育程度的分布情况(使用user_info)

作业3：2019年1月1日到2019年4月30日，每个时段的不同品类购买金额分布(使用user_trade)