一 内容回顾（列举前一天重点难点内容）

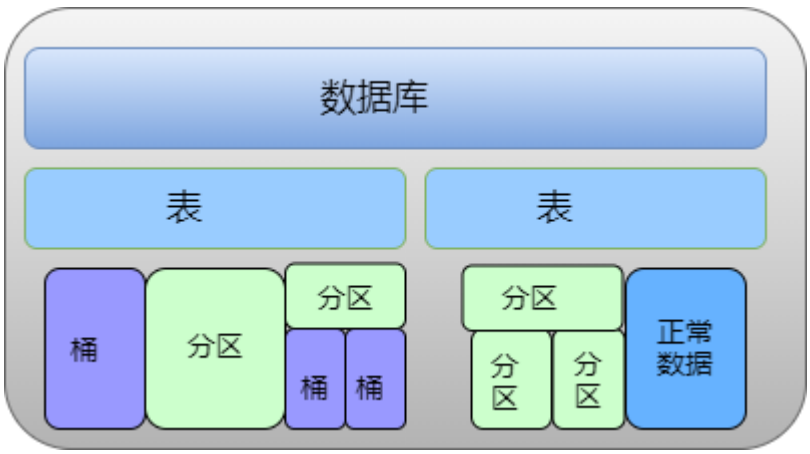
教学重点

```
Hive的简介
Hive的三种安装模式
表属性的修改
Hive的远程模式
Hive的架构
Hive的工作原理
Hive的基本操作
Hive加载数据的方式
表类型详解
分区表的相关概念
分区分类
分区操作
```

二 教学目标

```
分桶表来历
分桶表设置
抽样查询
分区分桶联合使用
Join连接
查询子句
排序
合并结果集
数据类型
系统内置函数
```

三 教学导读



3.1 为什么要分桶

```
当单个的分区或者表的数据量过大，分区不能更细粒度的划分数据，就需要使用分桶技术将数据划分成更细的粒度。
```

四 教学内容

4.1 分桶

4.1.1 分桶技术

```
[CLUSTERED BY (COLUMNNAME COLUMNTYPE [COMMENT 'COLUMN COMMENT'],...)
[SORTED BY (COLUMNNAME [ASC|DESC])...] INTO NUM_BUCKETS BUCKETS]
```

4.1.2 关键字及其原理

bucket

分桶的原理:跟MR中的HashPartitioner原理一样:都是key的hash值取模reduce的数量

- MR中：按照key的hash值除以reductTask取余
- Hive中：按照分桶字段的hash值去模除以分桶的个数

4.1.3 分桶的意义

```
为了保存分桶查询的分桶结构（数据按照分桶字段进行保存hash散列）
分桶表的数据进行抽样和JOIN时可以提高查询效率,一般是用来抽样查询
```

4.2 分桶表的创建示例

分桶表创建流程:

1. 首先要创建带分桶定义的表（分桶表）
2. 然后创建一个临时表（普通表）
3. 从临时表中使用分桶查询将查询到的数据插入到分桶表中

4.2.1 创建分桶表

```
create table if not exists buc1(
id int,
name string,
age int
)
clustered by (id) into 4 buckets
row format delimited fields terminated by ',';

数据：从(data/buc1.txt加载如下格式数据)
id,name,age
```

4.2.2 创建临时表

```
create table if not exists temp_buc1(
id int,
name string,
age int
)
row format delimited fields terminated by ',';
```

4.2.3 分桶使用load方式加载数据不能体现分桶

```
load data local inpath '/root/hivedata/buc1.csv' into table buc1;
```

4.2.4 加载数据到临时表

```
load data local inpath '/root/hivedata/buc1.csv' into table temp_buc1;
```

4.2.5 使用分桶查询将数据导入到分桶表

```
insert overwrite table buc1
select id,name,age from temp_buc1
cluster by (id);
```

这里还是无法分桶,需要进行下面的强制设置

4.2.6 设置强制分桶的属性

```
<!-- 如果要分桶,就要打开分桶的强制模式 -->
set hive.enforce.bucketing=false/true
<name>hive.enforce.bucketing</name>
<value>false</value>
<description>
  Whether bucketing is enforced. If true, while inserting into the table, bucketing is
enforced.</description>

最好把强制排序也打开
set hive.enforce.sorting=true;
```

4.2.7 如果设置了reduces的个数和总桶数不一样，请手动设置

```
set mapreduce.job.reduces=-1 #-1表示可以根据实际需要来定制reduces的数量
```

4.2.8 创建指定排序字段的分桶表

```
create table if not exists buc3(
id int,
name string,
age int
)
clustered by (id)
sorted by (id desc) into 4 buckets
row format delimited fields terminated by ',';
```

4.2.9 导入数据

```
insert overwrite table buc3
select id,name,age from temp_buc1
distribute by (id) sort by (id asc);
和下面的语句效果一样
insert overwrite table buc3
select id,name,age from temp_buc1
cluster by (id);

注意:定义和导数据时同时设置了分桶和排序,以导入数据时优先
```

4.3 分桶表查询案例

```
select * from buc3;
select * from buc3 tablesample(bucket 1 out of 1 on id);
```

4.3.1 查询第1桶的数据

```
语法解释:

tablesample(bucket x out of y on id)
x:代表从第几桶开始查询
y:可以是总桶数,总桶数的倍数或者是因子。x不能大于y。
y如果是4 ,已经有的总桶数是4    那么要取的数据 4/4= 1个桶
y如果是2 ,已经有的总桶数是4,那么要取的数据4/y 4/2=2个桶
on后面跟的是分桶的字段

  hive根据y的大小，决定抽样的比例。例如，table总共分了64份，当y=32时，抽取(64/32=)2个bucket的数据，当y=128时，抽取(64/128=)1/2个bucket的数据。

  x表示从哪个bucket开始抽取。例如，table总bucket数为32，tablesample(bucket 3 out of 16)，表示总共抽取（32/16=）2个bucket的数据，分别为第3个bucket和第（3+16=）第19个bucket的数据。

案例分析:
# 一共4桶,取得第一桶的数据
select * from buc3 tablesample(bucket 1 out of 4 on id);
# 取得第二桶的数据
#y如果是2 ,已经有的总桶数是4,那么要取的数据4/y 4/2=2个桶
select * from buc3 tablesample(bucket 2 out of 2 on id);
```

```
# 从x开始取,取的数量是 (4/2)=2,当x==1时代表取的是1,3桶
select * from buc3 tablesample(bucket 1 out of 2 on id);
# 4/y=1/2,代表每一桶被分成2份,当x=1和5时表示第一桶,x=1表示取的是第一桶的前半部分.
# 小技巧:区分id是那一部分的方法:id/y值,例如当前的案例:id=8,y=8,id/y=0,因为商+1=1=x,桶中的部分就是前半部分,比如:id=12,y=8,id/y=4,商+1=5=x,所以是桶的后半部分
select * from buc3 tablesample(bucket 1 out of 8 on id);
```

4.3.2 查询

（注意：tablesample一定是放在from后面）

查询id为基数：

```
select  *
from buc1 tablesample(bucket 2 out of 2 on id)
where name = "aa1";

select  *
from buc2
where uname = "aa1";

查询:
select * from temp_buc1 limit 3;
select * from temp_buc1 tablesample(3 rows);#行数
select * from temp_buc1 tablesample(30 percent);#数量的百分比
select * from temp_buc1 tablesample(6B);B k M G T P #具体的数量

SELECT * from temp_buc1 order by rand() limit 3; #rand()是随机排序
https://www.aboutyun.com/thread-27093-1-1.html
```

4.4 分区分桶联合案例

4.4.1案例

(先熟悉Select各种查询)

```
需求：按照性别分区（1男2女），在分区中按照id的奇偶进行分桶：

id,name,sex
数据请参考data/user.txt

建表:
create table if not exists stu(
id int,
name string
)
partitioned by (sex string)
clustered by (id) into 2 buckets
row format delimited fields terminated by ',';

建临时表:
```

```
create table if not exists stu_temp(
id int,
name string,
sex string
)
row format delimited fields terminated by ',';


-- 主要要把动态分区设为非严格模式,如下:
set hive.exec.dynamic.partition.mode=nonstrict;

-- 加载临时表的数据
load data local inpath '/root/hivedata/user.txt' into table stu_temp;


将数据导入到分区分桶表

insert overwrite table stu partition(sex)
select id,name,sex from stu_temp cluster by id ;

查询性别为女，学号为奇数的学生

select * from stu tablesample(bucket 2 out of 2 on id)
where sex = '2';
```

4.4.2 分桶表总结

```
1. 定义
    clustered by (id)         ---指定分桶的字段
    sorted by (id asc|desc)   ---指定数据的排序规则，表示咱们预期的数据是以这种规则进行的排序

2. 导入数据
    cluster by (id)   ---指定getPartition以哪个字段来进行hash，并且排序字段也是指定的字段，排序是以asc排列
    distribute by (id)    ---- 指定getPartition以哪个字段来进行hash
    sort by (name asc | desc) ---指定排序字段
    order by --- 全域排序(只能有一个reduce)

区别：cluster by 这种方式可以分别指定getPartition和sort的字段

导数据时：
    insert overwrite table buc3
    select id,name,age from temp_buc1
    distribute by (id) sort by (id asc);
    和下面的语句效果一样
    insert overwrite table buc4
    select id,name,age from temp_buc1
    cluster by (id) ;
```

4.4.3 注意事项

```
分区使用的是表外字段，分桶使用的是表内字段
分桶更加用于细粒度的管理数据，更多的是使用来做抽样、join
```

4.5 查询语句基本语法

4.5.1 select查询结构基本语法

下面是一个sql查询语句的基本结构

```
select selection_list # 查询的列
from table            # 要查询的表
join on               # 连接的表
where                 # 查询条件
group by              # 分组查询
having                # 分组条件过滤
order by              # 字段排序
sort by               # 结果排序
limit                 # 限制结果数
union/union all       # 合并表
```

4.5.2 sql语句的执行顺序

```
FROM
<left_table>
ON
<join_condition>
<join_type>
 JOIN
<right_table>
WHERE
<where_condition>
GROUP BY
<group_by_list>
HAVING
<having_condition>
SELECT
DISTINCT
<select_list>
ORDER BY
<order_by_condition>
LIMIT
<limit_number>
```

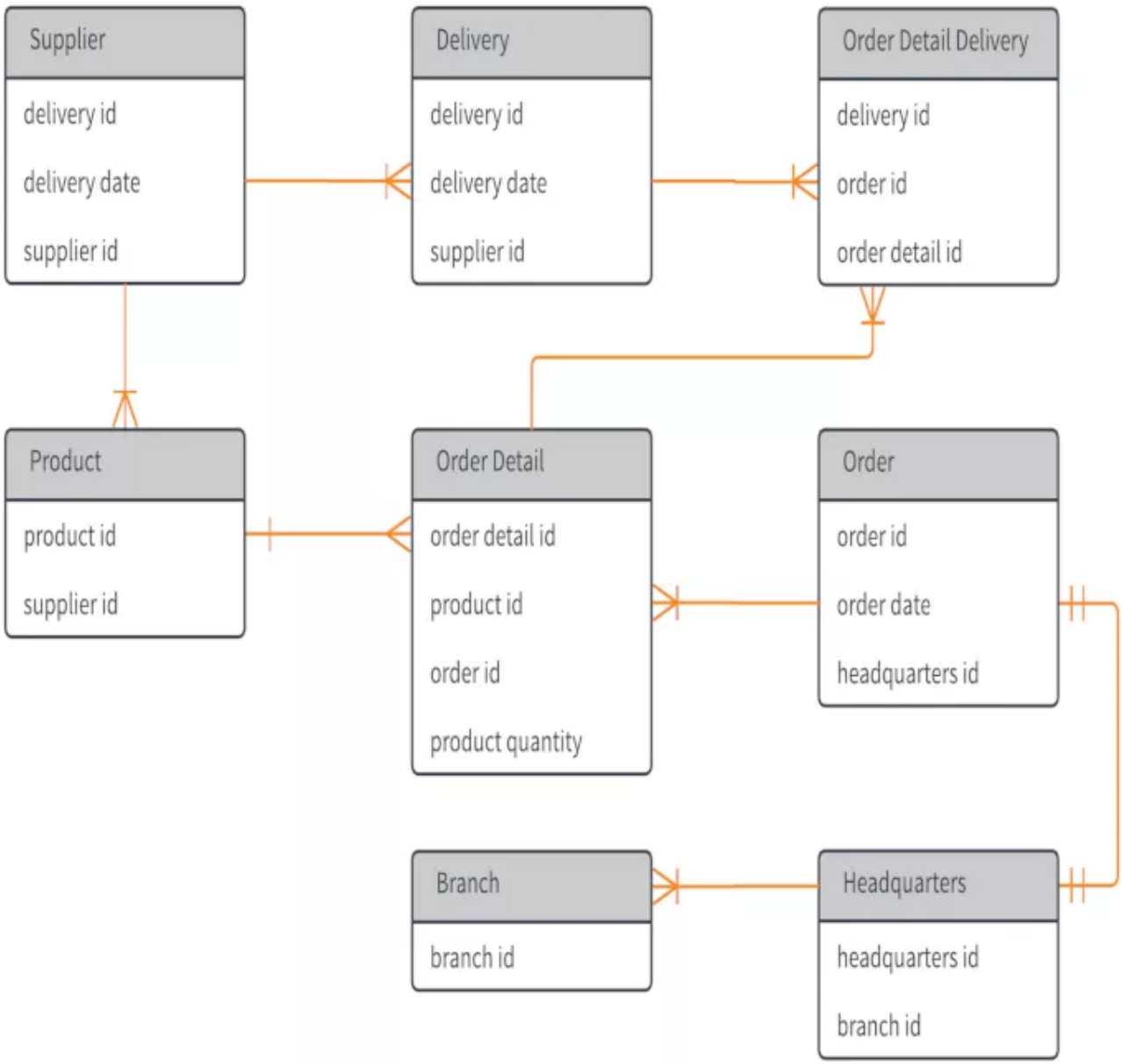
4.5.3 查询注意事项

```
尽量不要使用子查询、尽量不要使用 in not in
select * from aa1
where id in (select id from bb);
查询尽量避免join连接查询，但是这种操作咱们是永远避免不了的。
查询永远是小表驱动大表（永远是小结果集驱动大结果集）
```

4.5.4 数据库建模

关系型数据库最难的地方，就是建模（model）。

错综复杂的数据，需要建立模型，才能储存在数据库。所谓"模型"就是两样东西：实体（entity）+ 关系（relationship）ER图。实体指的是那些实际的对象，带有自己的属性，可以理解成一组相关属性的容器。关系就是实体之间的联系，通常可以分成"一对一"、"一对多"和"多对多"等类型。



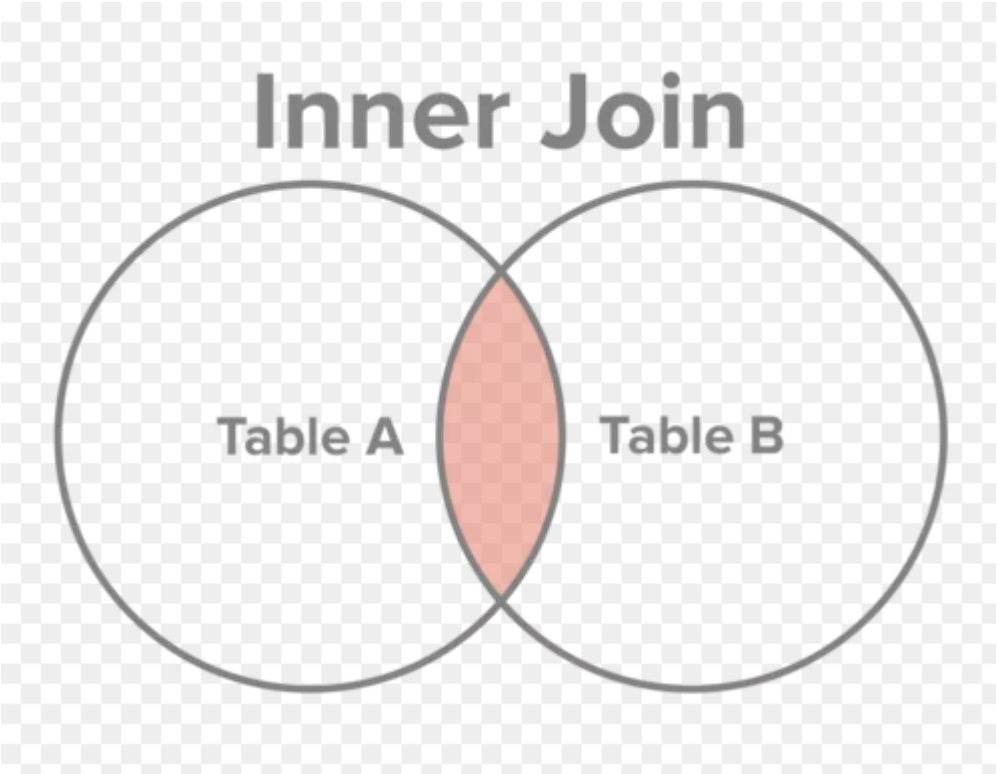
4.6 Join的语法与特点

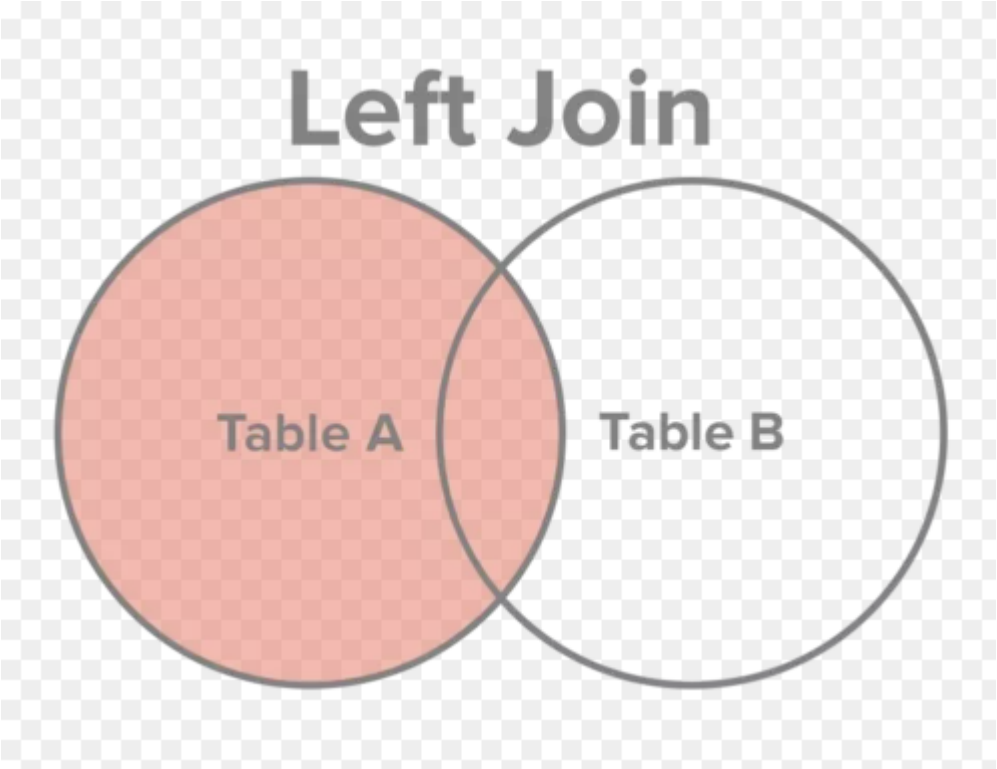
4.6.1 表之间关系

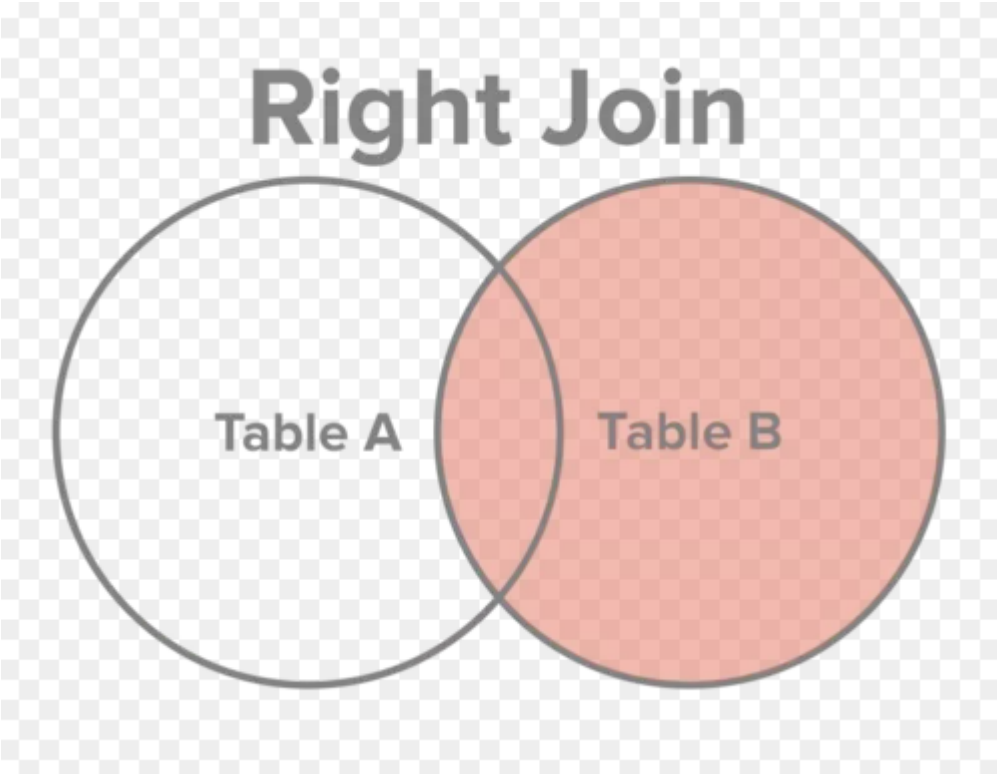
在关系型数据库里面，每个实体有自己的一张表（table），所有属性都是这张表的字段（field），表与表之间根据关联字段"连接"（join）在一起。所以，表的连接是关系型数据库的核心问题。

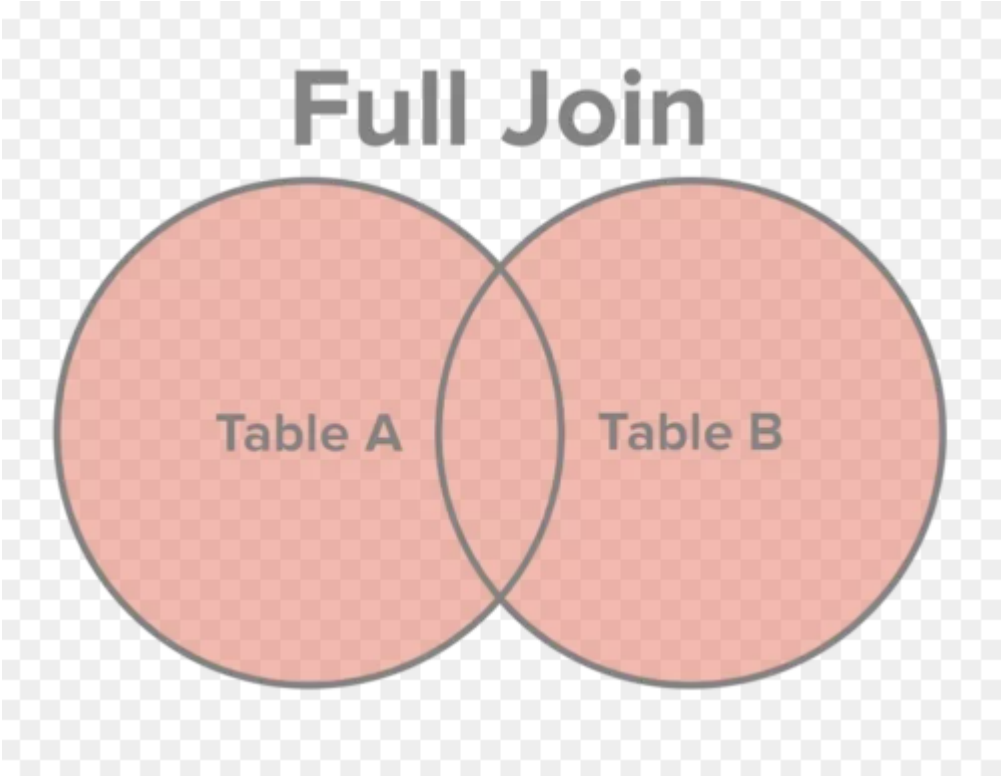
表的连接分成好几种类型。
内连接（inner join）
外连接（outer join）
左连接（left join）
右连接（right join）
全连接（full join）

以前，很多文章采用维恩图（两个圆的集合运算），解释不同连接的差异。




Left Join
Table A
Table B

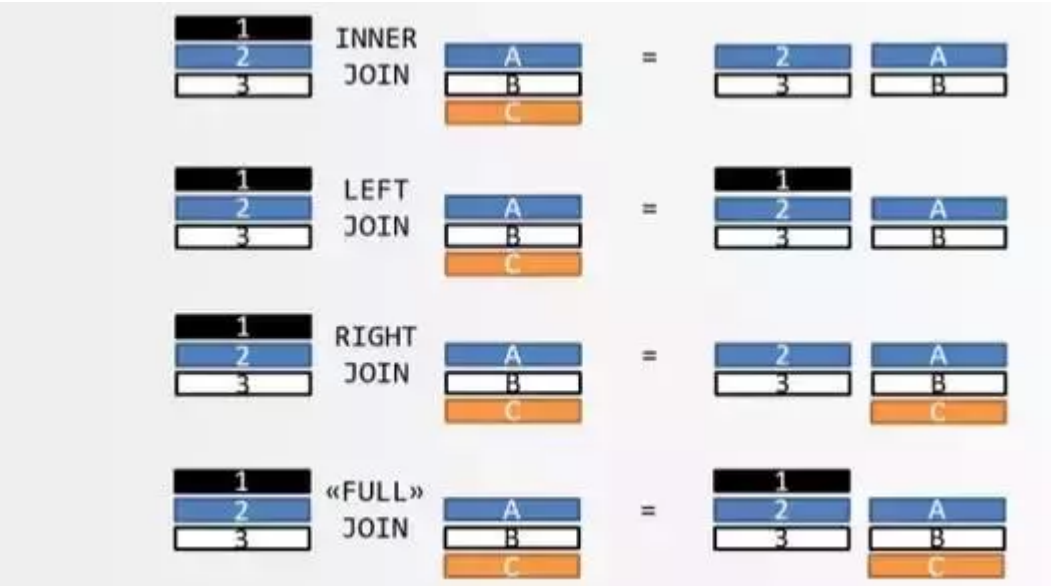

Right Join
Table A
Table B



所谓"连接"，就是两张表根据关联字段，组合成一个数据集。问题是，两张表的关联字段的值往往是不一致的，如果关联字段不匹配，怎么处理？比如，表 A 包含张三和李四，表 B 包含李四和王五，匹配的只有李四这一条记录。

很容易看出，一共有四种处理方法。
只返回两张表匹配的记录，这叫内连接（inner join）。
返回匹配的记录，以及表 A 多余的记录，这叫左连接（left join）。
返回匹配的记录，以及表 B 多余的记录，这叫右连接（right join）。
返回匹配的记录，以及表 A 和表 B 各自的多余记录，这叫全连接（full join）。

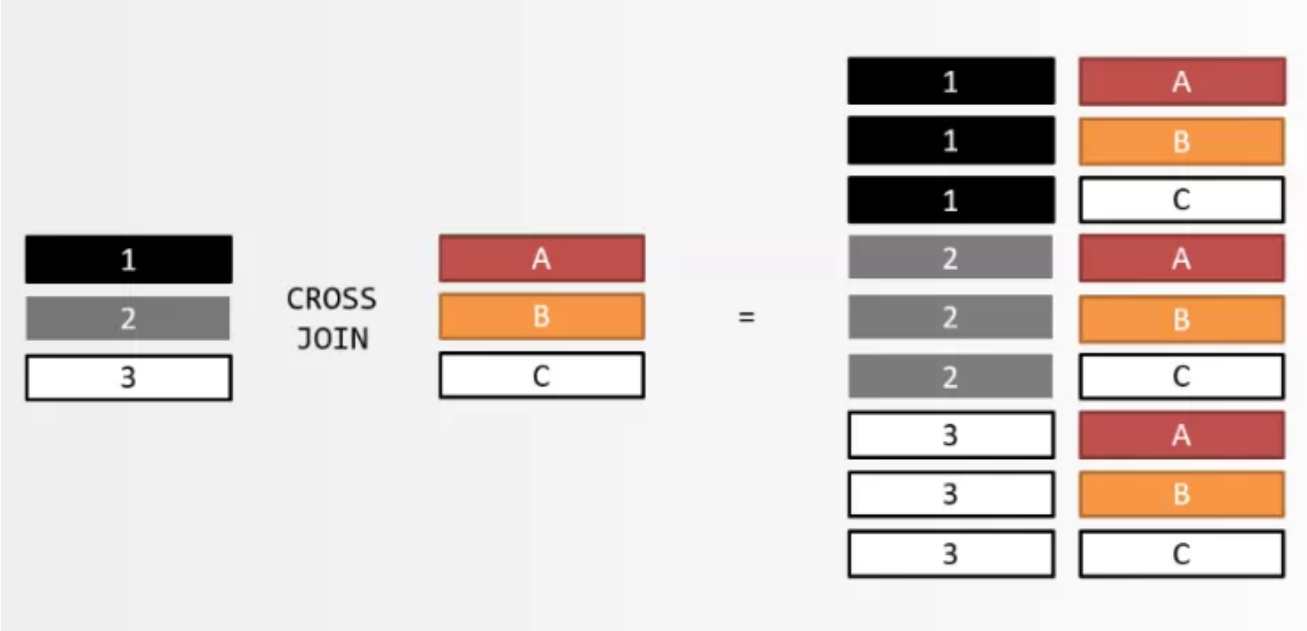
下图就是四种连接的图示。



上图中，表 A 的记录是 123，表 B 的记录是 ABC，颜色表示匹配关系。返回结果中，如果另一张表没有匹配的记录，则用 null 填充。

这四种连接，又可以分成两大类：内连接（inner join）表示只包含匹配的记录，外连接（outer join）表示还包含不匹配的记录。所以，左连接、右连接、全连接都属于外连接。

此外，还存在一种特殊的连接，叫做"交叉连接"（cross join），指的是表 A 和表 B 不存在关联字段，这时表 A（共有 n 条记录）与表 B （共有 m 条记录）连接后，会产生一张包含 n x m 条记录的新表（见下图）。



4.6.2 Join案例-语句特点演示

4.6.2.1 连接分类

```
类型：(left join\left outer join)
right join
right outer join
inner join
full outer join

特殊类型：
left semi join
```

准备数据

```
请参照/data/u1.txt ,/data/u2.txt
```

4.6.2.2 建两张表

```
create table if not exists u1(
id int,
name string
)
row format delimited fields terminated by ',';

create table if not exists u2(
id int,
name string
)
row format delimited fields terminated by ',';
```

4.6.2.3 加载数据

```
load data local inpath '/root/hivedata/u1.txt' into table u1;
load data local inpath '/root/hivedata/u2.txt' into table u2;
```

4.6.2.4 内连接

使用关键字join 、 inner join 、 多表用逗号分开

不加任何的on 或者是where过滤条件时，称之为笛卡尔积。

```
select *
from u1
join u2 on u1.id =u2.id;

select *
from u1
inner join u2 on u1.id =u2.id;

select *
from u1,u2
where u1.id=u2.id;
```

4.6.2.5 左连接

数据以左表的数据为准，左表存在的数据都查询出，右表的数据关联上就出来，关联不上以NULL代替

```
left join/left outer join/left semi join :

left join 从hive0.8版本开始有
left join 和 left outer join 几乎差不多
select *
from u1
left join u2 on u1.id =u2.id;

select *
from u1
left outer join u2 on u1.id =u2.id;

上面的 SQL 语句还可以加上where条件从句，对记录进行筛选，比如只返回表 A 里面不匹配表 B 的记录。
```

4.6.2.6 右连接

以右表为准，来匹配左表信息，如果匹配不上，使用NULL来代替。

```
right join /right outer join

select *
from u1
right join u2 on u1.id =u2.id;


select *
from u1
right outer join u2 on u1.id =u2.id;
hive不支持right semi join:
这两个得到的结果也是一样的。
```

4.6.2.7 全连接

```
full outer join
(相互进行连接，如果有一张表的数据连接不上来使用NULL来代替)
select *
from u1
full outer join u2 on u1.id =u2.id;

另一个例子，返回表 A 或表 B 所有不匹配的记录。
```

4.6.3 Hive专有Join特点

4.6.3.1 left semi join

在Hive中，有一种专有的join操作,left semi join,我们称之为半开连接。它是left join的一种优化形式，只能查询左表的信息，主要用于解决Hive中左表的数据是否存在的问题。相当于exists关键字的用法。

```
select *
from u1
left semi join u2 on u1.id =u2.id;

select u1.* ,u2.*
from u1
left semi join u2 on u1.id =u2.id;

select u1.*
from u1
where exists (select 1 from u2 where u2.id =u1.id);

# left semi join 叫做半开连接，通常是left join的一种优化，只能查询左表的信息，然后主要解决Hive中存在不存在的问题。
```

4.6.3.2 子查询

```
# 演示员工表
create table emp
(
    EMPNO    string,
    ENAME    string,
    JOB      string,
    MGR      string,
    HIREDATE string,
    SAL      string,
    COMM     string,
    DEPTNO   string
) row format delimited fields terminated by ','

# 演示部门表
create table dept
(
    DEPTNO    string,
    DNAME    string,
    LOC      string
) row format delimited fields terminated by ',';
```

准备数据:

请参照

```
data/emp.txt
data/dept.txt
```

导入数据:

```
load data local inpath  "/root/hivedata/emp.txt" into  table emp;
select * from emp;

load data local inpath  "/root/hivedata/dept.txt" into  table dept;
select * from dept;
```

```
# Hive对子查询支持不是很友好，特别是 "="问题较多
select
e.*
from emp e
where e.deptno = (
select
d.deptno
from dept d
limit 1
);

select
e.*
from emp e
where e.deptno in (
select
d.deptno
from dept d
);
```

```
# inner join 和outer join的区别：
# 分区字段对outer join 中的on条件是无效，对inner join 中的on条件有效

有inner join 但是没有full inner join
有full outer join但是没有outer join
所有join连接，只支持等值连接(= 和 and )。不支持 != 、 < 、> 、 <> 、>=、 <= 、or
```

4.6.3.3 map-side join

如果所有的表中有小表，将会把小表缓存内存中，然后在map端进行连接关系查找。Hive在map端 查找时将减小查询量，从内存中读取缓存小表数据，效率较快，还省去大量数据传输和shuffle耗时

```
#注意使用下面属性打开Map-Join：
set hive.auto.convert.join=true
select
e.*
from u1 d
join u2 e
on d.id = e.id;
```

```
# 以前的老版本，需要添加(/+MAPJOIN(小表名)/)来标识该join为map端的join。hive 0.7以后hive已经废弃，但是仍然管用：

select
/+MAPJOIN(d)/
e.*
from u1 d
join u2 e
on d.id = e.id
;
到底小表多大才会被转换为map-side join：
set hive.mapjoin.smalltable.filesize=25000000   约23.8MB
```

```
on ：  所有on只支持等值连接。

where ：  where后面通常是表达式 、还可以是非聚合函数表达式(但是不能是聚合函数表达式)

select
d.*
from dept d
where  length(d.dname) > 5 ;
```

4.6.3.4 表达式别名

对有些表达式来说,查询的结果字段名比较难理解,这时候可以给表达式起一个别名,方便查看结果的表头.使用as把复杂表达式命名为一个容易懂的别名

```
select
d.dname,
length(d.dname) as nameLength
from dept d ;
```

4.7 查询子句

4.7.1 Where语句特点

where后不能跟聚合函数

```
#下面代码不能正确执行
select
e.deptno,
count(e.deptno) ct
from emp e
where  count(e.deptno) > 3
group by e.deptno;
```

4.7.2 Group By语句特点

group by : 分组，通常和聚合函数搭配使用

查询的字段要么出现在group by 后面，要么出现在聚合函数里面

```
select
e.deptno,
count(e.ename) ct
from emp e
group by e.deptno;
```

4.7.3 Having语句

Hving是对分组以后的结果集进行过滤。

```
select
e.deptno,
count(e.deptno) ct
from emp e
group by e.deptno
having ct > 3;
```

4.7.4 Limit语句

限制从从结果集中取数据的条数,一般用于分页

```
# 将set hive.limit.optimize.enable=true 时，limit限制数据时就不会全盘扫描，而是根据限制的数量进行抽样。

同时还有两个配置项需要注意：
hive.limit.row.max.size         这个是控制最大的抽样数量
hive.limit.optimize.limit.file 这个是抽样的最大文件数量

select
    e.deptno,
    count(e.deptno) ct
from emp e
group by e.deptno
having ct > 3
limit 1;
```

4.7.5 排序

4.7.5.1 order by

```
全局排序:引发全表扫描,reduce数量一个,不建议使用
```

4.7.5.2 sort by

```
默认分区内排序,当reduceTask数量是1时候,那么效果和order by 一样
一般和distribute by 搭配使用
```

4.7.5.3 distribute by

```
用来确定用哪个列(字段)来分区,一般要写在sort by的前面
```

4.7.5.4 cluster by

```
分区的列和分区内排序的列相同时,那么可以用cluster by deptno来代替   distribute by deptno sort
cluster by : 兼有distribute by以及sort by的升序功能。
排序只能是升序排序（默认排序规则），不能指定排序规则为asc 或者desc。
```

4.7.5.5 排序详细区别

```
distribute by :  根据by后的字段和reducer个数，决定map的输出去往那个reducer。
默认使用查询的第一列的hash值来决定map的输出去往那个reducer。如果reducer的个数为1时没有任何体现。

sort by:局部排序，只保证单个reducer有顺序。
order by：全局排序，保证所有reducer中的数据都是有顺序。
如果reduser个数只有一个，两者都差不多。
两者都通常和 desc 、 asc 搭配。默认使用升序asc。

order by的缺点：
由于是全局排序，所以所有的数据会通过一个Reducer 进行处理，当数据结果较大的时候，一个Reducer 进行处理十分影响性能。
注意事项：
当开启MR 严格模式的时候ORDER BY 必须要设置 LIMIT 子句 ，否则会报错

-- 手动设置reducer个数：
set mapreduce.job.reduces=3;
select
e.empno
from emp e
order by e.empno desc ;

只要使用order by , reducer的个数将是1个。

```

4.8 表合并 Union

如果把两张表 结果联合在一起,可以用union,有下面两种用法

- union： 将多个结果集合并，去重，排序
- union all： 将多个结果集合并，不去重，不排序。

下面是具体代码演示

```
select
```

```
d.deptno as deptno,
d.dname as dname
from dept d
union
select
e.deptno as deptno,
e.ename as dname
from emp e;

select
d.deptno as deptno,
d.dname as dname
from dept d
union all
select
d.dname as dname,
d.deptno as deptno
from dept d;

-- 单个union 语句不支持：orderBy、clusterBy、distributeBy、sortBy、limit
-- 单个union语句字段的个数要求相同，字段的顺序要求相同。

如果想去重查询中的某列数据，可以使用 distinct关键字
```

4.9 数据类型

前面已经讲过基本数据类型,以下是复杂基本类型:

分类	类型	描述	字面量示例
复杂类型	ARRAY	有序的的同类型的集合	array(1,2)
	MAP	key-value,key必须为原始类型，value可以任意类型	map('a',1,'b',2)
	STRUCT	字段集合,类型可以不同	struct('1',1,1.0), named_stract('col1','1','col2',1,'clo3',1.0)
	UNION	在有限取值范围内的一个值	create_union(1,'a',63)

4.9.1 特殊基本类型

```
Java中有的而Hive中没有的：
long
char
short
byte
```

简单数据类型创表案例:

```
-- 创建一个函多种基本类型的表
create table if not exists bs1(
id1 tinyint,
id2 smallint,
id3 int,
id4 bigint,
sla float,
sla1 double,
isok boolean,
content binary,
dt timestamp
)
row format delimited fields terminated by ',';

-- 准备要插入的数据:
23, 12,342523,455345345,30000,600005,nihao, helloworld2 ,2017-06-02 11:41:30
12, 13,342526,455345346,80000,100000, true, helloworld1, 2017-06-02 11:41:30

load data local inpath '/root/hivedata/bs1.csv' into table bs1;
```

4.9.2 复杂的数据类型

复杂类型分为三种,分别是数组array,键值对map,和结构体struct

```
array : col array<基本类型> ,下标从0开始，越界不报错，以NULL代替
map   : column map<string,string>
struct: col struct
```

4.9.2.1 array示例

```
-- 数据如下：注意下面列之间是通过TAB来分隔的
zhangsan    78,89,92,96
lisi    67,75,83,94

# 注意terminated顺序,新建数组类型
create table if not exists arr1
(
    name  string,
    score array<string>
)
row format delimited fields terminated by '\t'
collection items terminated by ',';

# 导入数据
load data local inpath '/root/hivedata/arr1.csv' into table arr1;

# 查询:
select * from arr1;
select name,score[1] from arr1 where size(score) > 3;
```

4.9.2.1.1 列转行

就是把一列数据转化成多行,如下:

```
#原始数据:
zhangsan   90,87,63,76

#转化后数据
zhangsan        90
zhangsan        87
zhangsan        63
zhangsan        76
```

内嵌查询:

- **explode** : 展开

  列表中的每个元素生成一行

```
select explode(score) score from arr1;
```

- **lateral view** : 虚拟表

侧视图的意义是配合explode，一个语句生成把单行数据拆解成多行后的数据结果集。

```
select name,cj from arr1 lateral view explode(score) score as cj;
```

统计每个学生的总成绩:

```
select name,sum(cj) as totalscore from arr1 lateral view explode(score) score as cj group by
name;
```

4.9.2.1.2 行转列

就是把多行数据转化成一行数据:

```
#原始数据:
zhangsan        90
zhangsan        87
zhangsan        63
zhangsan        76

#转化后数据
zhangsan   90,87,63,76
```

- 准备数据:

```
create table arr_temp
as
select name,cj
from arr1 lateral view explode(score) score as cj;
```

- collect_set函数：

  它们都是将分组中的某列转为一个数组返回

```
create table if not exists arr3(
name string,
score array<string>
)
row format delimited fields terminated by ' '
collection items terminated by ',';
```

- 将数据写成array格式：

```
insert into arr3select name,collect_set(cj) from arr_temp group by name;
```

4.9.2.2 map示例

有数据如下:

```
zhangsan chinese:90,math:87,english:63,nature:76
lisi chinese:60,math:30,english:78,nature:0
wangwu chinese:89,math:25,english:81,nature:9
```

- 创建map类型的表

```
create table if not exists map1(
name string,
score map<string,int>
)
row format delimited fields terminated by ' '
collection items terminated by ','
map keys terminated by ':';
```

- 加载数据

```
load data local inpath '/root/hivedata/map1.txt' into table map1;
```

- map格式数据查询

```
#查询数学大于35分的学生的英语和自然成绩:
select
m.name,
m.score['english'] ,
m.score['nature']
from map1 m
where m.score['math'] > 35;
```

4.9.2.2.1 map列转行

使用上面的数据

- explode 展开

```
select explode(score) as (m_class,m_score) from map1;
```

- Lateral view

  Lateral View和split，explode等一起使用，它能够将一行数据拆成多行数据，并在此基础上对拆分后的数据进行聚合。

```
select name,m_class,m_score from map1 lateral view explode(score) score as m_class,m_score;
```

4.9.2.2.2 map行转列

准备数据:

```
name7,38,75,66
name6,37,74,65
name5,36,73,64
name4,35,72,63
name3,34,71,62
name2,33,70,61
name1,32,69,60
```

- 创建临时表,并加载数据

```
create table map_temp(
name string,
score1 int,
score2 int,
score3 int
)
row format delimited fields terminated by ',';
# map_temp.csv

load data local inpath '/root/hivedata/map_temp.csv' into table map_temp;
```

- 创建要导入数据map表

```
create table if not exists map2(
name string,
score map<string,int>
)
row format delimited fields terminated by ' '
collection items terminated by ','
map keys terminated by ':';
```

- 导入数据：

```
insert into map2
select name,map('chinese',score1,'math',score2,'english',score3) from map_temp;
```

4.9.2.3 struct

```
create table if not exists str1
(
name  string,
score struct<chinese:int,math:int,english:int>
)
    row format delimited fields terminated by '\t'
        collection items terminated by ',';
#导入数据：
load data local inpath '/opt/data/arr1.csv' into table str1;

查询数据：
# 查询数学大于35分的学生的英语和语文,数学成绩：
select name,
       score.english,
       score.chinese,
       score.math
from str1
where score.math > 35;
```

4.9.2.4 复杂数据类型案例

```
uid uname belong tax addr
1 xdd ll,lw,lg,lm wuxian:300,gongjijin:1200,shebao:300 北京,西城区,中南海
2 lkq lg,lm,lw,ll,mm wuxian:200,gongjijin:1000,shebao:200 河北,石家庄,中山路
```

```
#查询：下属个数大于4个，公积金小于1200，省份在河北的数据

create table if not exists tax(
id int,
name string,
belong array<string>,
tax map<string,double>,
```

```
addr struct<province:string,city:string,road:string>
)
row format delimited fields terminated by ' '
collection items terminated by ','
map keys terminated by ':'
stored as textfile;

# 导入数据

load data local inpath '/root/hivedata/tax.csv' into table tax;

#查询：下属个数大于4个，公积金小于1200，省份在河北的数据
select id,
name,
belong[0],
belong[1],
tax['wuxian'],
tax['shebao'],
addr.road
from tax
where size(belong) > 4 and
tax['gongjijin'] < 1200 and
addr.province = '河北';
```

4.9.2.5 嵌套数据类型(了解)

在Hive中,有时候数据结构比较复杂,这时候可以用到嵌套类型,就是一个类型里面可以放置另外一个类型.这时候为了避免数据发送混乱,一般可以使用Hive自带的分隔符,

Hive共支持8个层级的分隔符，依次是：\001,\002,\003,...\008

下面可以简单演示用一般分隔符来表示嵌套类型

```
-- map嵌套使用(数据为qt.txt)
uid uname belong tax addr
1   xdd wuxian:(300,300),gongjijin:1200,1500,shebao:300
2   lkq wuxian:(200,300),gongjijin:1000,1200,shebao:200

-- 定义嵌套类型数据表
create table qt(
id int,name string,
addr map<string,array<string>>)
row format delimited fields terminated by '\t'
collection items terminated by ','
map keys terminated by ':';

-- 加载数据类型,有些数据可能会丢失
load data local inpath '/root/hivedata/qt.txt' overwrite into table qt;
```

4.10 系统内置函数

4.10.1 函数查看

可以用下面两个命令查看Hive中的函数

```
-- 显示Hive中所有函数
show functions;

-- 查看某个函数的用法
desc function array;
```

4.10.2 日期函数

因为Hive的核心功能和海量数据统计分析,而在统计分析时日期时间是一个非常重要的维度, 所以日期函数在Hive使用中尤为重要.

```
-- 时间戳转日期
select from_unixtime(1505456567);
select from_unixtime(1505456567,'yyyyMMdd');
select from_unixtime(1505456567,'yyyy-MM-dd HH:mm:ss');

-- 获取当前时间戳
select unix_timestamp();

-- 日期转时间戳
select unix_timestamp('2017-09-15 14:23:00');

-- 计算时间差
select datediff('2018-06-18','2018-11-21');

-- 查询当月第几天
select dayofmonth(current_date);

-- 月末:
select  last_day(current_date);

--当月第1天:
select date_sub(current_date,dayofmonth(current_date)-1);
--下个月第1天:
select  add_months(date_sub(current_date,dayofmonth(current_date)-1),1);

-- 当前日期
select current_date

-- 字符串转时间（字符串必须为：yyyy-MM-dd格式）
select to_date('2017-01-01 12:12:12');

-- 日期、时间戳、字符串类型格式化输出标准时间格式:
select date_format(current_timestamp(),'yyyy-MM-dd HH:mm:ss');
select date_format(current_date(),'yyyyMMdd');
select date_format('2017-01-01','yyyy-MM-dd HH:mm:ss');
```

4.10.3 字符串函数

```
lower-- (转小写)
select lower('ABC');

upper-- (转大写)
select lower('abc');

length-- (字符串长度，字符数)
select length('abc');

concat-- (字符串拼接)
select concat("A", 'B');

concat_ws -- (指定分隔符)
select concat_ws('-','a' ,'b','c');

substr-- (求子串)
select substr('abcde',3);
```

4.10.4 类型转换函数

```
cast(value as type) -- 类型转换
select cast('123' as int)+1;
```

4.10.5 数学函数

```
round --四舍五入((42.3 =>42))
select round(42.3);

ceil --向上取整(42.3 =>43)
select ceil(42.3);

floor --向下取整(42.3 =>42)
select floor(42.3);
```

4.10.6 判断为空函数

```
nvl (expr1, expr2)
#作用：将查询为Null值转换为指定值。
#若expr1为Null，则返回expr2，否则返回expr1。
select nvl(count,2);
```

五 实战应用

5.1 实战案例一

六 教学总结

6.1 教学重点

```
分桶表设置
抽样查询
分区分桶联合使用
查询语句基本语法
数据库建模
表之间关系
Join连接
查询子句
排序
合并结果集
数据类型:array,map,struct
系统内置函数:常见函数
```

七 课后作业

7.1

创建一个分桶表,并且练习分桶表的数据插入:

答案:

```

```

7.2

练习分桶表的抽样查询,总共分8个桶,要求分三种情况抽样,可以取一桶,取4桶,取8桶的情况下,写出SQL语句

答案:

```

```

7.3

注意: 下面题目的数据参考Hive第一天的数据

1 查询男生、女生人数:

```

```

2 查询出只选修一门课程的全部学生的学号和姓名:

```
1
```

3 查询1981年出生的学生名单

```
1
```

4 查询平均成绩大于80的所有学生的学号、姓名和平均成绩：

```
1
```

5 查询每门课程的平均成绩，结果按平均成绩升序排序，平均成绩相同时，按课程号降序排列：

```
1
```

6 查询课程名称为“数学”，且分数低于60的学生名字和分数：

```
1
```

7 查询所有学生的选课情况：

```
1
```

8 查询任何一门课程成绩在70分以上的姓名、课程名称和分数：

```
1
```

9 查询01课程比02课程成绩高的所有学生的学号

```
1
```

10 查询平均成绩大于60分的同学的学号和平均成绩

```
1
```

11 查询所有同学的学号、姓名、选课数、总成绩

```
1
```

12 查询所有课程成绩小于60的同学的学号、姓名：

```
1
```

13 查询没有学全所有课的同学的学号、姓名：

```
1
```

14 查询至少有一门课与学号为01同学所学相同的同学的学号和姓名：

```
1
```

15 查询至少学过学号为01同学所有一门课的其他同学学号和姓名；

```
1
```

八 解决方案

8.1 应用场景

8.2 核心面试题