查询（单表 多表 子查询）

说在前面的话：在整个数据库的学习中，本模块对于新人来说是最难最繁琐的，无论是对先前学习的应用，还是接下来要学习很多的新关键字，都会极其繁琐的交织在一起。请放松心态，不要急于求成，查询是由很多个模块组成的（关键字），就好比玩游戏时要记住技能和技能的搭配方法，当你游戏入门了，也就是说这些关键字你掌握了，就不会感到厌烦，写起命令也会手到擒来。最后新人不要拿使用客户端去编辑SQL语句，客户端会有命令提示，习惯后面试会遭罪的。

单表查询

select

查询

专业术语：**要返回的列或表示式**

查表的数据内容

关于语法， select 后面接要查询的字段名（要查的字段名也可以有条件后面会说到）。

这里重复一下如果是查全表，不用吧每个字段写全，只需要写 * 星号就可以。如果是多个字段，每个字段之间用逗号隔开。

计算

select 还可以用于简单计算

MySQL 支持的算术运算符

运算符	作用
+	加法
-	减法
*	乘法
/ 或 DIV	除法
% 或 MOD	取余

在除法运算和模运算中，如果除数为0，将是非法除数，返回结果为NULL。

例：

```
mysql> select 2*3+4/2;
+---------+
| 2*3+4/2 |
+---------+
|  8.0000 |
+---------+
1 row in set (0.01 sec)
```

from

专业术语：**从中检索数据的表**

在查询语句中配合 select 来确定取那张表里的字段，所以 from 后接表名。

这里说一下，笛卡尔积（这东西现实中真的很难去用到）。

简单说就是我们在查询的时候，在 from 后查询的表名为多个时，数据库会将这些表拼接起来进行显示，拼接顺序见例

例：新建两张表并插入数据

```
mysql> select * from t01;
+----+-------+
| id | name  |
+----+-------+
|  1 | jerry |
|  2 | tom   |
|  3 | jack  |
+----+-------+
3 rows in set (0.00 sec)

mysql> select * from t02;
+------------+------------+
| number_one | number_two |
+------------+------------+
|         15 |         20 |
|        328 |        876 |
+------------+------------+
2 rows in set (0.00 sec)
```

笛卡尔积

```
mysql> select * from `t01`,`t02`;
+----+-------+------------+------------+
| id | name  | number_one | number_two |
+----+-------+------------+------------+
|  1 | jerry |         15 |         20 |
|  1 | jerry |        328 |        876 |
|  2 | tom   |         15 |         20 |
|  2 | tom   |        328 |        876 |
|  3 | jack  |         15 |         20 |
|  3 | jack  |        328 |        876 |
+----+-------+------------+------------+
6 rows in set (0.00 sec)
```

as

别名

为了优化查询时的可读性，我们可以给表和字段起一个别名，但别名不能在除查询以外的情况下使用。

表别名

给表起别名真的很难用到但是了解写法和注意事项

```
`表名` as `别名`
```

这里的　as　其实是可以省略的，如要省略需要将表名和别名用空格隔开。（不推荐）

注意：表的别名不能与该数据库的其它表同名。

字段别名

```
字段名 as 别名
```

这里的　as　其实是可以省略的，如要省略需要将表名和别名用空格隔开。（不推荐）

注意：字段的别名不能与该表的其它字段同名。在条件表达式中不能使用字段的别名，否则会出现“ERROR 1054 (42S22): Unknown column”这样的错误提示信息。

例：

```
mysql> select * from `student`;
+----+----------+--------+---------------+
| id | name     | gender | student_phone |
+----+----------+--------+---------------+
|  1 | Tom      | 1      | 暂时未知      |
|  2 | Tom      | 1      | 123456789     |
|  4 | Disapain | 2      | 122222222     |
|  5 | SANDEAO  | 2      | 12222222233   |
|  6 | Sitelin  | 2      | 12222223344   |
+----+----------+--------+---------------+
5 rows in set (0.00 sec)

mysql> select name as student_name, id as student_id from `student`;
+--------------+------------+
| student_name | student_id |
+--------------+------------+
| Tom          |          1 |
| Tom          |          2 |
| Disapain     |          4 |
| SANDEAO      |          5 |
| Sitelin      |          6 |
+--------------+------------+
5 rows in set (0.00 sec)
```

扩展

as可以作为连接语句的操作符。

```
create table  tablename2 as select * from tablemate1;
```

解释：上面语句的意思就是先获取到 tablemate1 表中的所有记录，之后创建一张 tablemate2 表，结构和 tablemate1 表相同，记录为后面语句的查询结果。(效果为复刻一张结构和数据一摸一样的表，可以用作备份)

也可以在本库备份其他库的表，只需要在 from 库名.表名 即可。（该操作可变向改库名）

也可在后面接查询语句，把查出来的虚表变成实表保存起来。

dual

虚拟的表名

在实际写代码中dual是可以省略的。如果在实际应用中遇到特殊的要求：select 后面必须匹配 from 的时候可以使用。

例如

```
select 7*9 from dual;      计算器
SELECT SYSDATE() from dual      获取系统时间
```

---

where

专业术语：行级过滤（以实表为主开展查询）

在查**表**内字段数据时，对数据进行筛选 （筛选语句会用到：**比较运算符** 和 **逻辑运算符** 可根据自己需要进行搭配）

```
where 字段 筛选语句;
```

例：我们使用 != and not in() 搭配来查一张表

```
mysql> select * from `fruit_order` where id !=4 and order_id not in(10);
+----------+----------+------+
| order_id | money    | id   |
+----------+----------+------+
|        1 | 111.1100 |   10 |
|        2 | 222.2200 |    2 |
|        5 | 555.5000 |   10 |
|        6 | 212.3100 |    2 |
|        9 | 345.1200 |   10 |
+----------+----------+------+
5 rows in set (0.00 sec)
```

比较运算符

符号	描述	备注
=	等于	
<>, !=	不等于	
>	大于	
<	小于	
<=	小于等于	
>=	大于等于	
BETWEEN	在两值之间	>=min&&<=max
NOT BETWEEN	不在两值之间	
IN()	在集合中	
NOT IN()	不在集合中	
<=>	严格比较两个NULL值是否相等	两个操作码均为NULL时，其所得值为1；而当一个操作码为NULL时，其所得值为0
LIKE	模糊匹配	
REGEXP 或 RLIKE	正则式匹配	
IS NULL	为空	将表中空数据条筛选出来
IS NOT NULL	不为空	将表中非空数据筛选出来

逻辑运算符

运算符号	作用
NOT 或 !	逻辑非
AND	逻辑与 （必须满足所有条件）
OR	逻辑或 （满足一个条件即可）
XOR	逻辑异或

---

where中对运算符的应用

常见的运算符就不细说了看图表就可以了，下面说一些，MySQL特色的运算符。

in与not in

in 在集合中

not in 不在集合中

这两个运算符呈现对立关系

```
where 字段 in('筛选数据','筛选数据');
```

解析：筛选字段数据中符合筛选数据的数据。如果数据是字段别忘单引号。如果是not in 就是不符合排除法。我们称之为非连续性范围。

例：我们进行多字段多数据排除筛选

```
mysql> select * from `fruit_order` where id not in(4,2) and order_id not in(10,5);
+----------+----------+------+
| order_id | money    | id   |
+----------+----------+------+
|        1 | 111.1100 |   10 |
|        9 | 345.1200 |   10 |
+----------+----------+------+
2 rows in set (0.00 sec)
```

between与not between

在两值之间

不在两者之间

```
where 字段 between '筛选数据' and '筛选数据';
```

解析：筛选范围内的数据，and 左右为范围。**注意:实际筛选出来的数据会包括范围数据。**

例：

```
mysql> select * from `fruit_order` where order_id between 5 and 10;
+----------+-----------+------+
| order_id | money     | id   |
+----------+-----------+------+
|        5 |  555.5000 |   10 |
|        6 |  212.3100 |    2 |
|        7 | 2212.3000 | NULL |
|        8 |  256.3900 |    4 |
|        9 |  345.1200 |   10 |
|       10 | 1231.2400 |    2 |
+----------+-----------+------+
6 rows in set (0.01 sec)
```

like

模糊查询 模糊匹配

查询不确定因素，但是这些因素有共同点。

```
where 字段 like '模糊的筛选数据和配合使用的筛选符号';
```

注意：配合使用的筛选符号和模糊筛选数据之间不能有空格。

配合like语句使用的符号

% 表示匹配一个或多个任意字符，可放在模糊筛选数据的前面后面连边甚至中间。

%放在后面：只要开头含有筛选数据将被筛选出。

%放在后面：只要结尾含有筛选数据的将被筛出。

以此类推，%在两边就是含有中间将筛出，%在中间就是两边含有将筛出

例：喜闻乐见水果订单表，我们显示全部数据。

```
+----------+-----------+------+
| order_id | money     | id   |
+----------+-----------+------+
|        1 |   111.1100 |   10 |
|        2 |   222.2200 |    2 |
|        3 |   333.3330 | NULL |
|        4 |   444.4400 |    4 |
|        5 |   555.5000 |   10 |
|        6 |   212.3100 |    2 |
|        7 |  2212.3000 | NULL |
|        8 |   256.3900 |    4 |
|        9 |   345.1200 |   10 |
|       10 |  1231.2400 |    2 |
|       11 |   983.4500 | NULL |
|       12 | 87345.0000 |    4 |
|       13 |   234.4321 |    2 |
|       14 |   196.4321 |   10 |
+----------+-----------+------+
14 rows in set (0.00 sec)
```

我们来查询这张表中money这列数据有关%在1各种位置的查询效果。

```
mysql> select * from `fruit_order` where money like '1%';
+----------+----------+------+
| order_id | money    | id   |
+----------+----------+------+
|        1 |  111.1100 |   10 |
|       10 | 1231.2400 |    2 |
|       14 |  196.4321 |   10 |
+----------+----------+------+
3 rows in set (0.00 sec)
```

```
mysql> select * from `fruit_order` where money like '%1';
+----------+----------+------+
| order_id | money    | id   |
+----------+----------+------+
|       13 | 234.4321 |    2 |
|       14 | 196.4321 |   10 |
+----------+----------+------+
2 rows in set (0.00 sec)
```

```
mysql> select * from `fruit_order` where money like '%1%';
+----------+----------+------+
| order_id | money    | id   |
+----------+----------+------+
|        1 |  111.1100 |   10 |
|        6 |  212.3100 |    2 |
|        7 | 2212.3000 | NULL |
|        9 |  345.1200 |   10 |
|       10 | 1231.2400 |    2 |
|       13 |  234.4321 |    2 |
|       14 |  196.4321 |   10 |
+----------+----------+------+
7 rows in set (0.00 sec)
```

```
mysql> select * from `fruit_order` where money like '1%1';
+----------+----------+------+
| order_id | money    | id   |
+----------+----------+------+
|       14 | 196.4321 |   10 |
+----------+----------+------+
1 row in set (0.00 sec)
```

_（下划线） 表示匹配一个任意字符

一个下划线表示一个字符两个下划线表示两个字符,可写多个下划线。

```
where 字段 like '模糊筛选数据_';
```

注意：下划线的摆放位置同上面%的位置一样，**但是使用下划线查询是由字符限制，只能查到下划线加模糊筛选数据总和字符数的数据。**

例：为了查到111.1100，使用该方法是不好的，这里演示该方法的局限性，但是用这个方法查名字或短而精确的数据还是很有用的。

```
mysql> select * from `fruit_order` where money like '1_';
Empty set (0.00 sec)

mysql> select * from `fruit_order` where money like '11_100';
Empty set (0.00 sec)

mysql> select * from `fruit_order` where money like '11__100';
Empty set (0.00 sec)

mysql> select * from `fruit_order` where money like '11___100';
+----------+----------+------+
| order_id | money    | id   |
+----------+----------+------+
|        1 | 111.1100 |   10 |
+----------+----------+------+
1 row in set (0.00 sec)
```

聚合函数

作用于结构表中一行或者多行，最终返回一个结果，用作统计使用。

大白话就是给每个字段里的数据做各种统计。

sum 求和

avg 平均值

max 最大值

min 最小值

count 计算个数

```
select  聚合函数(字段) from `表名`
```

---

order by 排序查询

专业术语：**输出排序顺序**

把所要查的数据进行排序（可以筛选后排序，也可以不筛选直接排序）

升序关键字　　**asc**

降序关键字　　**desc**

当不写关键字默认为升序

```
order by 字段名 升序或降序
```

解析：该语句要放在整体语句的最后。

例：不筛选直接排序

```
mysql> select * from `fruit_order`order by money asc;
+----------+------------+------+
| order_id | money      | id   |
+----------+------------+------+
|        1 |   111.1100 |   10 |
|       14 |   196.4321 |   10 |
|        6 |   212.3100 |    2 |
|        2 |   222.2200 |    2 |
|       13 |   234.4321 |    2 |
|        8 |   256.3900 |    4 |
|        3 |   333.3330 | NULL |
|        9 |   345.1200 |   10 |
|        4 |   444.4400 |    4 |
|        5 |   555.5000 |   10 |
|       11 |   983.4500 | NULL |
|       10 |  1231.2400 |    2 |
|        7 |  2212.3000 | NULL |
|       12 | 87345.0000 |    4 |
+----------+------------+------+
14 rows in set (0.00 sec)
```

查特定字段，起别名，筛选，升序排序。

```
mysql> select money AS order_money from `fruit_order` where money <1000 order by money asc;
+-------------+
| order_money |
+-------------+
|    111.1100 |
|    196.4321 |
|    212.3100 |
|    222.2200 |
|    234.4321 |
|    256.3900 |
|    333.3330 |
|    345.1200 |
|    444.4400 |
|    555.5000 |
|    983.4500 |
+-------------+
11 rows in set (0.00 sec)
```

查特定字段，起别名，筛选，降序排序。

```
mysql> select money AS order_money from `fruit_order` where money <1000 order by money desc;
+-------------+
| order_money |
+-------------+
|    983.4500 |
|    555.5000 |
|    444.4400 |
|    345.1200 |
|    333.3330 |
|    256.3900 |
|    234.4321 |
|    222.2200 |
|    212.3100 |
|    196.4321 |
|    111.1100 |
+-------------+
11 rows in set (0.01 sec)
```

group by分组查询

专业术语：**分组说明**

用于统计时进行分组，必须配合聚合函数使用。

```
select 聚合函数查询的字段名,分组的字段名 from `表名` group by 分组的字段名;
```

解析：上面这条语法大白话就是：去查询数据（select 聚合函数查询的字段名,分组的字段名），这些数据来自与哪里（from `表名`）。通过那个字段进行分组（group by 分组的字段名）

注意

1.group by 后面必须是你要分组的字段名。分组的字段名也要必须在 select 查询的内容中。聚合函数字段，和分组的字段名没有顺序要求。

2.group by 必须配合聚合函数或函数使用，所以大多情况下，我们分组的都是和数字有关。

3.group by 可以加入筛选语句，可以先分组再筛选，也可以先筛选再分组，或者一起用也是可以的，根据实际情况筛选语句放在group by 前和后。**注意：分组前筛选用 where,分组后筛选我们 having。**

4.group by 分组查询后 **分组的字段名** 显示默认为升序，如需降序可在后面添加 desc。如果想排聚合函数查询的字段名还时要使用order by。（详细看例子）

例：还是水果订单案例，这回咱们根据水果种类，去查在整张订单表各种类水果共花了多少钱。

聚合函数求和字段为money 分组字段为id

分组的字段名 显示默认为升序（这里起了别名关注表中水果名编号）

```
mysql> select sum(money) as '水果进货花费' , id as '水果名编号' from `fruit_order` group by id ;
+--------------+------------+
| 水果进货花费 | 水果名编号 |
+--------------+------------+
|    3529.0830 |       NULL |
|    1900.2021 |          2 |
|   88045.8300 |          4 |
|    1208.1621 |         10 |
+--------------+------------+
4 rows in set (0.00 sec)
```

分组的字段名 显示改为降序（这里起了别名关注表中水果名编号）

```
mysql> select sum(money) as '水果进货花费' , id as '水果名编号' from `fruit_order` group by id desc;
+--------------+------------+
| 水果进货花费 | 水果名编号 |
+--------------+------------+
|    1208.1621 |         10 |
|   88045.8300 |          4 |
|    1900.2021 |          2 |
|    3529.0830 |       NULL |
+--------------+------------+
4 rows in set, 1 warning (0.00 sec)
```

聚合函数查询的字段名（这里起了别名关注表中水果进货花费）排升序使用 order by

```
mysql> select sum(money) as '水果进货花费' , id as '水果名编号' from `fruit_order` group by id order by money desc;
+--------------+------------+
| 水果进货花费 | 水果名编号 |
+--------------+------------+
|   88045.8300 |          4 |
|    3529.0830 |       NULL |
|    1900.2021 |          2 |
|    1208.1621 |         10 |
+--------------+------------+
4 rows in set (0.00 sec)
```

跟group by有关的扩展

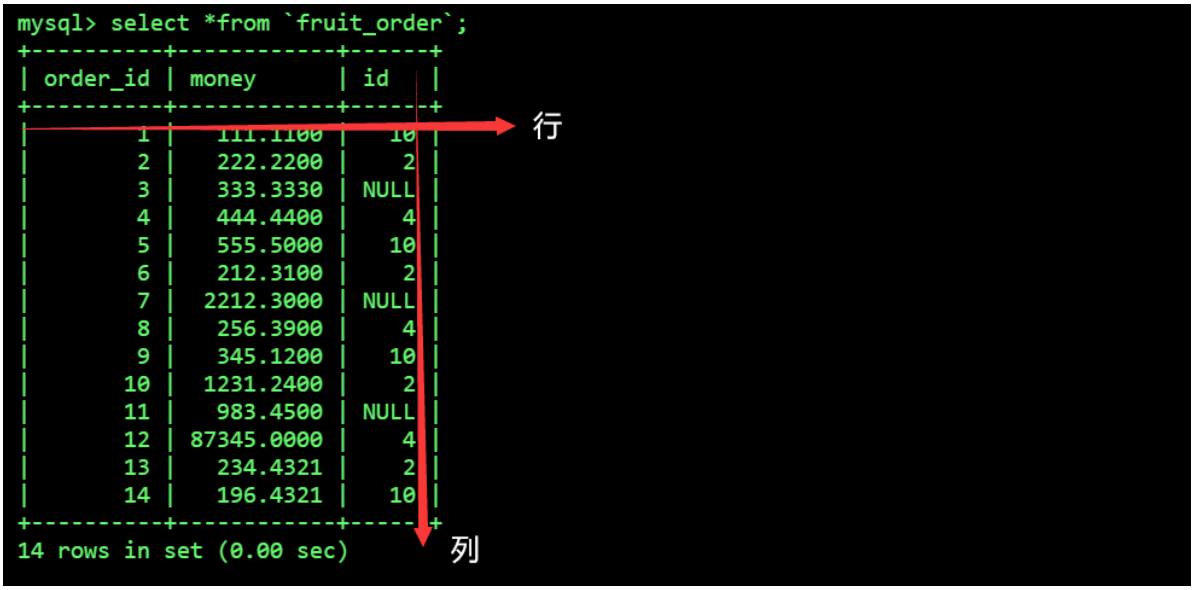
group_concat（）

将group by产生的同一个分组中的值连接起来，返回一个字符串结果每一个数据通过逗号隔开。

例：查每个水果种类花的每笔钱。

```
mysql> select group_concat(money) , id as '水果名编号' from `fruit_order` group by id ;
+-----------------------------------+------------+
| group_concat(money)               | 水果名编号 |
+-----------------------------------+------------+
| 333.3330,2212.3000,983.4500       |       NULL |
| 222.2200,212.3100,1231.2400,234.4321 |          2 |
| 444.4400,256.3900,87345.0000      |          4 |
| 111.1100,555.5000,345.1200,196.4321 |         10 |
+-----------------------------------+------------+
4 rows in set (0.00 sec)
```

原表



having

专业术语：**组级过滤**（以虚表为主展开查询）

负责查询后的再次筛选。

```
select 聚合函数查询的字段名,分组的字段名 from `表名` group by 分组的字段名 having  聚合函数查询的字段名 筛选语句;
```

注意：我们是在筛选后的表（虚表）上接着筛选，所以在使用having的使用字段名如果有别名要使用别名。

例：

```
mysql> select sum(money) as '水果进货花费' , id as '水果名编号' from `fruit_order` group by id having 水果进货花费<5000;
+--------------+------------+
| 水果进货花费 | 水果名编号 |
+--------------+------------+
|    3529.0830 |       NULL |
|    1900.2021 |          2 |
|    1208.1621 |         10 |
+--------------+------------+
3 rows in set (0.00 sec)

mysql> select sum(money) as '水果进货花费' , id as '水果名编号' from `fruit_order` group by id having 水果名编号<2;
Empty set (0.00 sec)

mysql> select sum(money) as '水果进货花费' , id as '水果名编号' from `fruit_order` group by id having 水果名编号>2;
+--------------+------------+
| 水果进货花费 | 水果名编号 |
+--------------+------------+
|   88045.8300 |          4 |
|    1208.1621 |         10 |
+--------------+------------+
2 rows in set (0.00 sec)
```

```
mysql> select * from `fruit_order` where order_id<10 having money<500;
+----------+----------+------+
| order_id | money    | id   |
+----------+----------+------+
|        1 | 111.1100 |   10 |
|        2 | 222.2200 |    2 |
|        3 | 333.3330 | NULL |
|        4 | 444.4400 |    4 |
|        6 | 212.3100 |    2 |
|        8 | 256.3900 |    4 |
|        9 | 345.1200 |   10 |
+----------+----------+------+
7 rows in set (0.00 sec)
```

---

limit

专业术语：要检索的行数

大白话：查表时按照设定显示表内指定的记录数（几条数据）。

```
limit 整数,整数;
```

注意：

1.limit后的参数必须是一个整数常量。

2.设定两个参数中间用逗号隔开，用于确定范围。实际显示范围是从第一个参数+1处开始和第二参数为显示几条数据。

3.当设定参数只有一个数的时候，作用为显示几条数据。

4.limit在实表和虚表都可以使用，根据实际情况确定好。

例：

设定两个参数，用于虚表。（还是上面order by 分组案例）

```
mysql> select sum(money),id from `fruit_order`group by id limit 1,3;
+------------+------+
| sum(money) | id   |
+------------+------+
|  1900.2021 |    2 |
| 88045.8300 |    4 |
|  1208.1621 |   10 |
+------------+------+
3 rows in set (0.00 sec)
```

设定一个参数，用于虚表。

```
mysql> select sum(money),id from `fruit_order`group by id limit 3;
+------------+------+
| sum(money) | id   |
+------------+------+
|  3529.0830 | NULL |
|  1900.2021 |    2 |
| 88045.8300 |    4 |
+------------+------+
3 rows in set (0.00 sec)
```

设定两个参数，用于实表。（原表为order by 原表）

```
mysql> select * from `fruit_order` limit 3,10;
+----------+------------+------+
| order_id | money      | id   |
+----------+------------+------+
|        4 |   444.4400 |    4 |
|        5 |   555.5000 |   10 |
|        6 |   212.3100 |    2 |
|        7 |  2212.3000 | NULL |
|        8 |   256.3900 |    4 |
|        9 |   345.1200 |   10 |
|       10 |  1231.2400 |    2 |
|       11 |   983.4500 | NULL |
|       12 | 87345.0000 |    4 |
|       13 |   234.4321 |    2 |
+----------+------------+------+
10 rows in set (0.00 sec)
```

distinct

用来去除查询结果中的重复记录（去重）

这个语句可在`select`、`insert`、`delete`和`update`中只可以在`select`中使用

```
select distinct 字段名 from tables;
```

注意：使用时一定要贴近修饰字段，否则会出现失效情况。详见案例。

例：查表有几个不同地区，需要去重再统计数量。

```
mysql> select * from student;
+----+------+------+--------+----------+---------+
| id | name | age  | gender | address  | phone   |
+----+------+------+--------+----------+---------+
|  1 | 刘一 |   11 | 0      | shanghai | 11111   |
|  2 | 陈二 |   12 | 0      | beijing  | 22222   |
|  3 | 张三 |   11 | 1      | gansu    | unknown |
|  4 | 李四 |   13 | 0      | shanghai | 44444   |
|  5 | 王五 |   11 | 1      | shanghai | 213123  |
+----+------+------+--------+----------+---------+
5 rows in set (0.00 sec)

mysql> select distinct (count(address)) as `有几个地区` from student;
+------------+
| 有几个地区 |
+------------+
|          5 |
+------------+
1 row in set (0.00 sec)

mysql> select count(distinct(address)) as `有几个地区` from student;
+------------+
| 有几个地区 |
+------------+
|          3 |
+------------+
1 row in set (0.00 sec)
```

注意 distinct 的位置导致查询结果的不同

单表查询结束

子句	说明	是否必须使用
select	要返回的列或表示式	是
form	从中检索数据的表	仅在从表选择数据时使用
where	行级过滤	否
group by	分组说明	仅在按组计算聚集时使用
having	组级过滤	否
order by	输出排序顺序	否
limit	要检索的行数	否

多表查询

又叫多表连接，就是将几张表拼接为一张表，然后进行查询。

union

连接两个以上的 SELECT 语句的结果组合到一个结果集合中

大白话：union意思是联盟联合的意思，正如union翻译的意思一样union连接的表简单粗暴（就像上世纪某个红色帝国一样），只是把数据单纯的拼在一起杂乱。

```
select 字段名,字段名,字段名 from `表名` where 筛选条件
union all或distinct或不写
select 字段名,字段名,字段名 from `表名` where 筛选条件;
```

注意：

1.在 union 后的**distinct:** 删除结果集中重复的数据。默认情况下 UNION 操作符已经删除了重复数据，所以 DISTINCT 修饰符对结果没啥影响，可写可不写。

2.在 union 后的**all**：返回所有结果集，包含重复数据。如果不想去重，显示全部的数据需要把all带上。

3。两张表在连接时字段数量必须相同，否则会报错。

例：这回我们将使用新的案例，学生表和成绩表。

学生表

```
mysql> select * from `student`;
+----+-----+-------+--------+----------------+
| id | age | name  | gender | stuPhone       |
+----+-----+-------+--------+----------------+
|  1 |  18 | tom   | 1      | 1145141919810  |
|  2 |  20 | jerry | 1      | 11451419198101 |
|  3 |  20 | sam   | 0      | 11451419198102 |
|  4 |  20 | rose  | 0      | 11451419198103 |
|  5 |  21 | jack  | 1      | 11451419198105 |
|  6 |  19 | tom   | 1      | 11451419198106 |
|  7 |  18 | rose  | 0      | 11451419198107 |
+----+-----+-------+--------+----------------+
7 rows in set (0.00 sec)
```

成绩表

```
mysql> select * from `score`;
+----+-------+-------+---------+---------+
| id | stuid | math  | chinese | english |
+----+-------+-------+---------+---------+
|  1 |     1 | 99.00 |   95.50 |   97.00 |
|  2 |     2 |  NULL |   92.00 |   94.50 |
|  3 |     3 |  NULL |    NULL |    NULL |
|  4 |     4 | 84.50 |   88.00 |   87.00 |
|  5 |     5 | 88.50 |   82.00 |   95.00 |
|  6 |     6 | 96.00 |   94.00 |   72.50 |
|  7 |     7 | 45.50 |   99.00 |   99.00 |
|  8 |     8 |  NULL |   99.00 |   99.00 |
+----+-------+-------+---------+---------+
8 rows in set (0.00 sec)
```

这两张表字段数量相同，所以我们可以直接用*不用写字段，进行链接

```
mysql> select * from `student` union all select * from `score`;
+----+-----+-------+--------+----------------+
| id | age | name  | gender | stuPhone       |
+----+-----+-------+--------+----------------+
|  1 |  18 | tom   | 1      | 1145141919810  |
|  2 |  20 | jerry | 1      | 11451419198101 |
|  3 |  20 | sam   | 0      | 11451419198102 |
|  4 |  20 | rose  | 0      | 11451419198103 |
|  5 |  21 | jack  | 1      | 11451419198105 |
|  6 |  19 | tom   | 1      | 11451419198106 |
|  7 |  18 | rose  | 0      | 11451419198107 |
|  1 |   1 | 99.00 | 95.50  | 97.00          |
|  2 |   2 | NULL  | 92.00  | 94.50          |
|  3 |   3 | NULL  | NULL   | NULL           |
|  4 |   4 | 84.50 | 88.00  | 87.00          |
|  5 |   5 | 88.50 | 82.00  | 95.00          |
|  6 |   6 | 96.00 | 94.00  | 72.50          |
|  7 |   7 | 45.50 | 99.00  | 99.00          |
|  8 |   8 | NULL  | 99.00  | 99.00          |
+----+-----+-------+--------+----------------+
15 rows in set (0.00 sec)
```

总结：显示结果来看这种方法是有局限性的（真就是起到简单的连接作用），尤其时对第二张表的可读性问题（第二张表没有任何字段标注），只适合同类型的表之间作连接。

---

inner join

内连接

前面说的union，只是简单的拼数据，实际上当我们要把两张表和在一起查询肯定是有目的，如果还是用union将丝毫没有帮助，这时我们可以使用inner join。

inner join 可将两张表连接的同时，还能通过设置共同参考字段（作用与外键相似，但是实际不同，可以理解为两个字段匹配一下），使两张表的一些数据发生关系，并在查询输出时合并成一张表，来满足我们对多元信息的查询要求。（结合下面案例理解会更好）

```
select A表字段,b表字段 from `A表名` inner join `b表名`on `A表名` A表名.A表关系字段= b表名.b表关系字段;
```

注意：select 后的字段可根据自己需要写多个。

例：我们将学生表中的姓名与学生成绩表中各科目进行内连接，把学生表的 id 和成绩表的 stuid 设定为共用字段。

```
mysql> select name,math,chinese,english  from `student` inner join `score` on student.id= score.stuid;
+-------+-------+---------+---------+
| name  | math  | chinese | english |
+-------+-------+---------+---------+
| tom   | 99.00 |   95.50 |   97.00 |
| jerry |  NULL |   92.00 |   94.50 |
| sam   |  NULL |    NULL |    NULL |
| rose  | 84.50 |   88.00 |   87.00 |
| jack  | 88.50 |   82.00 |   95.00 |
| tom   | 96.00 |   94.00 |   72.50 |
| rose  | 45.50 |   99.00 |   99.00 |
+-------+-------+---------+---------+
7 rows in set (0.00 sec)
```

left join 和right join

左连接和右连接

大白话:以左表为基准进行查询和以右表为基准进行查询

```
select A表字段,b表字段 from `A表名` left或right join `b表名`on A表名.A表关系字段= b表名.b表关系字段;
```

例：

我们将学生表与学生成绩表进行左连接，把学生表的 id 和成绩表的 stuid 设定为共用字段。

```
mysql> select name,math,chinese,english from `student` left join `score` on student.id= score.stuid;
+-------+-------+---------+---------+
| name  | math  | chinese | english |
+-------+-------+---------+---------+
| tom   | 99.00 |   95.50 |   97.00 |
| jerry |  NULL |   92.00 |   94.50 |
| sam   |  NULL |    NULL |    NULL |
| rose  | 84.50 |   88.00 |   87.00 |
| jack  | 88.50 |   82.00 |   95.00 |
| tom   | 96.00 |   94.00 |   72.50 |
| rose  | 45.50 |   99.00 |   99.00 |
+-------+-------+---------+---------+
7 rows in set (0.00 sec)
```

我们将学生表与学生成绩表进行右连接，把学生表的 id 和成绩表的 stuid 设定为共用字段。

```
mysql> select name,math,chinese,english from `student` right join `score` on student.id= score.stuid;
+-------+-------+---------+---------+
| name  | math  | chinese | english |
+-------+-------+---------+---------+
| tom   | 99.00 |   95.50 |   97.00 |
| jerry |  NULL |   92.00 |   94.50 |
| sam   |  NULL |    NULL |    NULL |
| rose  | 84.50 |   88.00 |   87.00 |
| jack  | 88.50 |   82.00 |   95.00 |
| tom   | 96.00 |   94.00 |   72.50 |
| rose  | 45.50 |   99.00 |   99.00 |
| NULL  |  NULL |   99.00 |   99.00 |
+-------+-------+---------+---------+
8 rows in set (0.00 sec)
```

总结：由上例我们发现

1.我们在使用inner join 的时候是默认为左表连接

2.左右表如果数据没有匹配，数据库会自动补null（案例中成绩表有第八个成绩然而学生表没有第八个学生）

cross join

交叉连接

这个关键字在实际应用中很难使用到。

```
select * from `A表名` cross join `b表名`;
```

这个关键字很尴尬，如果后面不加筛选条件将返回显示笛卡尔积，加了筛选条件返回显示和内连接功能相同。

```
mysql> select name,math from `student` cross join `score` on student.id=score.stuid;
+-------+-------+
| name  | math  |
+-------+-------+
| tom   | 99.00 |
| jerry |  NULL |
| sam   |  NULL |
| rose  | 84.50 |
| jack  | 88.50 |
| tom   | 96.00 |
| rose  | 45.50 |
+-------+-------+
7 rows in set (0.00 sec)
```

natural join

自然连接

数据库自己去找两张表内相同字段名进行内连接。

```
select * from `A表名` natural join `b表名`;
```

注意：

1.因为不可以设置非相同字段名进行行关系匹配，所以如果使用该方法，两张表的字段名一定要设定好。

2.该方法也可以使用左连接和右连接，在natural和join，中间加上 左：left 右：right 即可。

3.如果两张表中没有共同字段返回显示笛卡尔积

4.当出现多个相同名字字段时，我们可以在后面加入using()，在括号里填我们要连接的字段。

```
select * from `A表名` natural join `b表名` using(连接的字段);
```

多表查询部分结束

我们在实际使用中多数使用 inner join 虽然语句长但是写的全关系好辩认，有更好的可读性

子查询

子查询是将一个查询语句嵌套再另一个查询语句中的查询方式（不局限于 where 后也可以在from 后 select 后 having后等等）

大白话：select语句套娃查询。

```
select * from `A表名` where 根据条件选怎合适的运算符 (select 与A表有联系的b表字段 from `b表名` where 筛选条件或接着套娃);
```

特点：

1.子查询的内层查询结果，可以作为外层查询语句提供查询条件

2.子查询中可以包含`IN`、`NOT IN`、`AND`、`ALL`、`EXISTS`、`NOT EXISTS`等关键字

3.子查询中还可以包含比较运算符，如`=`、`!=`、`>`、`<`等

exists 与 not exists

存在与不存在

当查询的这张表中，存在筛选条件的数据时，将表中所有的数据列出。

当查询的这张表中，不存在筛选条件的数据时，将表中所有的数据列出。

例：依旧是学生表和成绩表，这次我们查语文成绩小于80分是否存在，存在列出表不存在列出表俩种筛选方式。

成绩表

```
mysql> select * from `score`;
+----+-------+-------+---------+---------+
| id | stuid | math  | chinese | english |
+----+-------+-------+---------+---------+
|  1 |     1 | 99.00 |   95.50 |   97.00 |
|  2 |     2 |  NULL |   92.00 |   94.50 |
|  3 |     3 |  NULL |    NULL |    NULL |
|  4 |     4 | 84.50 |   88.00 |   87.00 |
|  5 |     5 | 88.50 |   82.00 |   95.00 |
|  6 |     6 | 96.00 |   94.00 |   72.50 |
|  7 |     7 | 45.50 |   99.00 |   99.00 |
|  8 |     8 |  NULL |   99.00 |   99.00 |
+----+-------+-------+---------+---------+
8 rows in set (0.00 sec)
```

查询

```
mysql> select * from `student` where not exists (
    -> select stuid from `score` where chinese<80);
+----+-----+-------+--------+----------------+
| id | age | name  | gender | stuPhone       |
+----+-----+-------+--------+----------------+
|  1 |  18 | tom   | 1      | 1145141919810  |
|  2 |  20 | jerry | 1      | 11451419198101 |
|  3 |  20 | sam   | 0      | 11451419198102 |
|  4 |  20 | rose  | 0      | 11451419198103 |
|  5 |  21 | jack  | 1      | 11451419198105 |
|  6 |  19 | tom   | 1      | 11451419198106 |
|  7 |  18 | rose  | 0      | 11451419198107 |
+----+-----+-------+--------+----------------+
7 rows in set (0.00 sec)

mysql> select * from `student` where exists (
    -> select stuid from `score` where chinese<80);
Empty set (0.00 sec)
```

由此成绩表可见没有语文成绩没有小于80的，同样是语文成绩小于80，not exists 显示表 exists不显示表。

查询的内容结束

多练习，多实战，都不是圣人，都会忘。最后附上查询语句的编写顺序，和执行顺序。

编写顺序

select 字段名

from 表1 inner|left|right join 表2 on 表1 与表2 的关系...

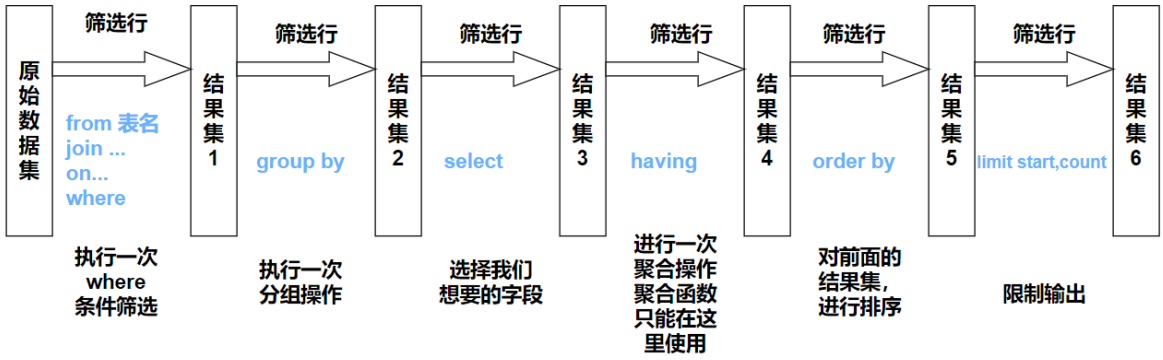
where ...

group by ... having ...

order by

limit start, count;

执行顺序



多种查询联合使用

水果表和水果订单表

```
mysql> select * from fruit;
+----+------------+-----------+
| id | name       | in_origin |
+----+------------+-----------+
|  1 | banana     | 云南      |
|  2 | mango      | 海南      |
|  3 | pineapple  | 海南      |
|  4 | strawberry | 河北      |
|  5 | pear       | 新疆      |
+----+------------+-----------+
5 rows in set (0.01 sec)

mysql> select *from t01;
+-------+----+-------+------------+
| order | id | level | money      |
+-------+----+-------+------------+
|     1 |  2 |     5 |   1499.670 |
|     2 |  4 |     5 |   1009.670 |
|     3 |  1 |     5 |  10009.670 |
|     4 |  3 |     5 |   1145.140 |
|     5 |  5 |     5 |  19198.100 |
|     6 |  1 |    10 |  15198.100 |
|     7 |  2 |    15 | 155198.100 |
|     8 |  3 |     4 |  78251.150 |
|     9 |  4 |    40 | 478251.150 |
|    10 |  5 |    10 | 796345.150 |
+-------+----+-------+------------+
10 rows in set (0.00 sec)
```

案例一

联合查询+分组查询+虚表查询+别名+排序+聚合函数

```
mysql> select name as `水果名` ,sum(money) as `单个水果总和` from fruit left join t01 on fruit.id=t01.id group by name ha
ving `单个水果总和`>50000 order by `单个水果总和`desc limit 3;
+------------+--------------+
| 水果名     | 单个水果总和 |
+------------+--------------+
| pear       |   815543.250 |
| strawberry |   479260.820 |
| mango      |   156697.770 |
+------------+--------------+
3 rows in set (0.00 sec)
```

解析：把每种水果的订单钱数总和进行筛选，其中查出总和大于五万的前三名，并输出名字和总和钱数。

案例二

子查询在from 后

由于我们使用 group by分组查询使用聚合函数。

```
mysql> select name ,sum(level) as level ,sum(money) as money from fruit right join t01 on fruit.id=t01.id group by name h
aving level>10 order by money desc limit 3;
+------------+-------+------------+
| name       | level | money      |
+------------+-------+------------+
| pear       |    15 | 815543.250 |
| strawberry |    45 | 479260.820 |
| mango      |    20 | 156697.770 |
+------------+-------+------------+
3 rows in set (0.00 sec)
```

但我就想只输出名字我们可以在此条名另外再套一层查询，直接查询这个虚表，但是**注意了要给这个虚表起别名。**

```
mysql> select name from(select name ,sum(level) as level ,sum(money) as money from fruit right join t01 on fruit.id=t01.i
d group by name having level>10 order by money desc limit 3) as `销冠`;
+------------+
| name       |
+------------+
| pear       |
| strawberry |
| mango      |
+------------+
3 rows in set (0.01 sec)
```

别忘给虚表起别名