子查询：sub query，查询是在某个查询结果之上进行的，一条select语句内部包含了另外一条select语句。

分类

子查询有两种分类方式，分别为：按结果分类和按位置分类。

按结果分类，即根据子查询得到的数据进行分类（理论上，任何一个查询结果都可以理解为一个二维表），分别为：

标量子查询：子查询得到的结果是一行一列，出现的位置在where之后；

列子查询：子查询得到的结果是一列多行，出现的位置在where之后；

行子查询：子查询得到的结果是多行一列（多行多列），出现的位置在where之后；

表子查询：子查询得到的结果是多行多列，出现的位置在from之后。

按位置分类，即根据子查询（select语句）在外部查询（select语句）中出现的位置进行分类，分别为：

from子查询：子查询出现在from之后；

where子查询：子查询出现在where条件之中；

exists子查询：子查询出现在exists里面。

标量子查询

需求：现知道班级名称为PM3.1，想要获取该班的全部学生。

思路：

先确定数据源，学生表。

select \* from student where c\_id = ?;

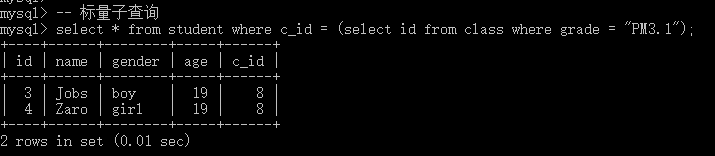
然后获取班级 ID，可以通过（班级表）班级名称来确定。

select id from class where grade = "PM3.1";

执行如下 SQL 语句，进行测试：

-- 标量子查询

select \* from student where c\_id = (select id from class where grade = "PM3.1");

[](https://camo.githubusercontent.com/aaaee023da2ca7008e6d74d052965b52136dab01/687474703a2f2f696d672e626c6f672e6373646e2e6e65742f3230313730383236323233313038393134)

列子查询

需求：查询所有在读班级（学生表中存在的班级）的学生。

思路：

先确定数据源，学生表。

select \* from student where c\_id in ?;

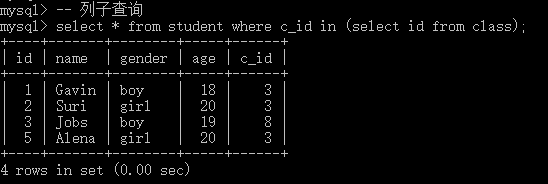
然后确定全部有效的班级 ID。

select id from class;

执行如下 SQL 语句，进行测试：

-- 列子查询

select \* from student where c\_id in (select id from class);

[](https://camo.githubusercontent.com/93591037acd6f18462ecddab4086fa2e02a36897/687474703a2f2f696d672e626c6f672e6373646e2e6e65742f3230313730383236323331363535373834)

如上图所示，我们完成了列子查询。在列子查询的结果为一行多列时，我们需要使用in作为条件进行匹配；此外，在 MySQL 中还有三个类似的条件，分别为：all、some和any。

any等价于in，表示其中一个；

any等价于smoe，而any和some用于否定时却有些区别；

all表示等于全部。

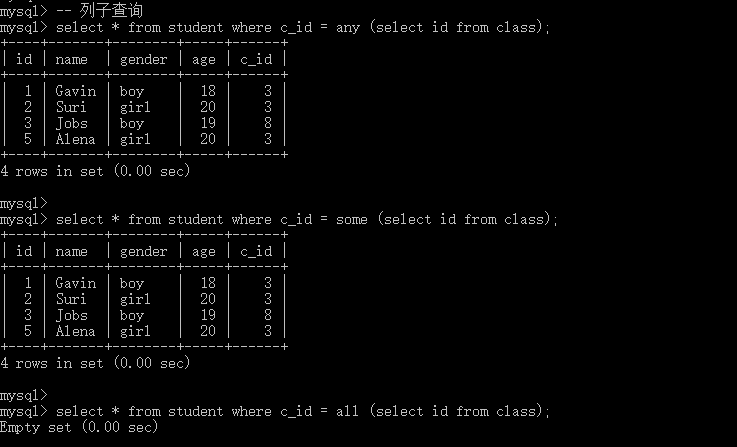
值得注意的是，在我们使用上面三个关键字中任何一个的时候，都需要搭配=使用，例如：

-- 列子查询

select \* from student where c\_id = any (select id from class);

select \* from student where c\_id = some (select id from class);

select \* from student where c\_id = all (select id from class);

[](https://camo.githubusercontent.com/b8ffc045e7b3d387904a99f1b5151257111e5ae8/687474703a2f2f696d672e626c6f672e6373646e2e6e65742f3230313730383236323331363136363435)

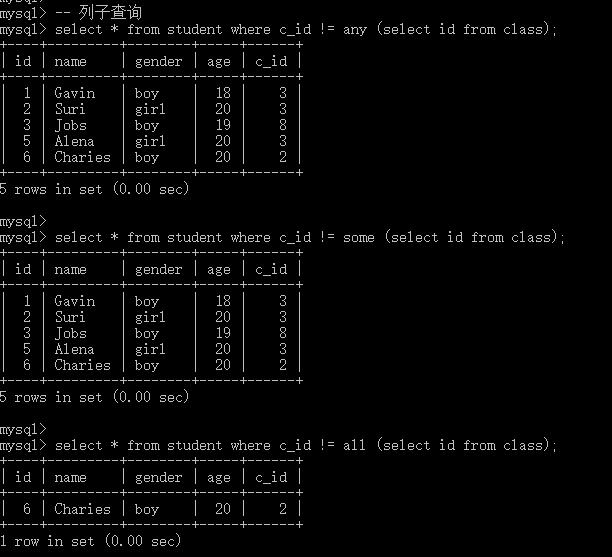
如上图所示，为any、some和all的肯定用法，下面我们来测试其否定用法：

-- 列子查询

select \* from student where c\_id != any (select id from class);

select \* from student where c\_id != some (select id from class);

select \* from student where c\_id != all (select id from class);

[](https://camo.githubusercontent.com/f930cb2341b63c58897bc562d7ee537907a4f332/687474703a2f2f696d672e626c6f672e6373646e2e6e65742f3230313730383236323331353236373837)

观察上图，我们会发现any、some和all在用于否定时，其会将null值排除掉。实际上，在真正的开发中，这三个关键字并不常用。

行子查询

行子查询，返回的结果可以使一行多列或者多行多列。

需求：查询学生表中，年龄最大且身高最高的学生。

思路：

先确定数据源，学生表。

select \* from student where age = ? and height = ?;

然后确定最大年龄和最大身高。

select max(age), max(height) from student;

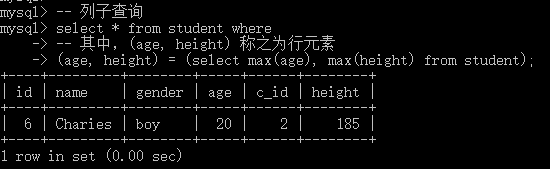
执行如下 SQL 语句，进行测试：

-- 列子查询

select \* from student where

-- 其中，(age, height) 称之为行元素

(age, height) = (select max(age), max(height) from student);

[](https://camo.githubusercontent.com/f32527b2d6d22d3cf70c0519d96c847607180890/687474703a2f2f696d672e626c6f672e6373646e2e6e65742f3230313730383237313035393339303532)

表子查询

表子查询，返回的结果是多行多列二维表（将子查询的结果当做二维表来使用），实际上，任何查询返回的结果都可以称之为二维表。

需求：找出每个班身高最高的学生。

思路：

先确定数据源，将学生按身高进行降序排序。

select \* from student order by height desc;

从每个班级选出第一个学生。

select \* from student group by c\_id;

在这里，我们可能会有些疑问：为什么要将学生表降序排序？为什么从每个班级选出第一个学生就可以？这是因为group by（分组）只会取表中分组字段的第一条记录，而当我们将学生表按身高降序排序时，（每组）身高最高的学生就会出现在第一位。

执行如下 SQL 语句，进行测试：

-- 表子查询

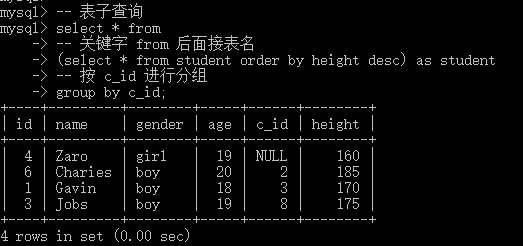
select \* from

-- 关键字 from 后面接表名

(select \* from student order by height desc) as student

-- 按 c\_id 进行分组

group by c\_id;

[](https://camo.githubusercontent.com/c408dec756239d49736d0f34e75c7070881e1a9d/687474703a2f2f696d672e626c6f672e6373646e2e6e65742f3230313730383237313132333336383333)

由上面的 SQL 语句可知，表子查询也是from子查询，即有select语句位于from之后。

exists子查询

exists：表示是否存在的意思，因此exists子查询就是用来判断某些条件是否满足（跨表），exists是接在where之后，其返回的结果为1或0，满足条件为1，反之为0.

需求：在班级存在的前提下，查询所有的学生。

思路：

先确定数据源。

select \* from student where ?;

然后确定条件是否满足。

exists(select \* from class);

执行如下 SQL 语句，进行测试：

-- exists 子查询

select \* from student where

exists(select \* from class);

-- 添加限定条件，满足条件

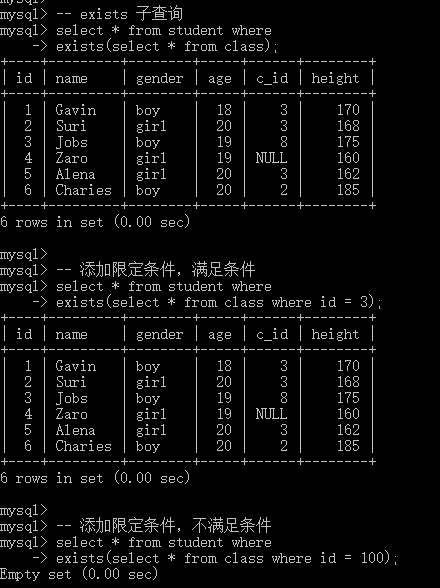
select \* from student where

exists(select \* from class where id = 3);

-- 添加限定条件，不满足条件

select \* from student where

exists(select \* from class where id = 100);

[](https://camo.githubusercontent.com/3ebaaf169ec291a2ff42b3a4ff8f8f92e561da9f/687474703a2f2f696d672e626c6f672e6373646e2e6e65742f3230313730383237313230343537353331)

至此，我们已经将子查询学习完啦！也许大家还会有些疑惑，那就是到底在什么时候用什么子查询？对于这个问题，我们不用过于纠结，因为这根本就是我们要用什么子查询的问题，而是根据实际需求，我们将查询返回的结果按形式命名的称呼而已。