

20000字干货笔记，一天搞定Mysql~

原创 远辰 数据不吹牛 1周前

↑ 关注 + 星标 ~ 别错过我噢~



今天给大家分享的是我学习Mysql记录的详细笔记，有基础知识，也有实战案例，文章较长，建议收藏~

基本语法

```
1  -- 显示所有数据库
2  show databases;
3
4  -- 创建数据库
5  CREATE DATABASE test;
6
7  -- 切换数据库
8  use test;
9
10 -- 显示数据库中的所有表
11 show tables;
12
13 -- 创建数据表
14 CREATE TABLE pet (
15     name VARCHAR(20),
16     owner VARCHAR(20),
17     species VARCHAR(20),
18     sex CHAR(1),
19     birth DATE,
20     death DATE
21 );
22
23 -- 查看数据表结构
```



```
24 -- describe pet;
25 desc pet;
26
27 -- 查询表
28 SELECT * from pet;
29
30 -- 插入数据
31 INSERT INTO pet VALUES ('puffball', 'Diane', 'hamster', 'f', '1990-03-30');
32
33 -- 修改数据
34 UPDATE pet SET name = 'squirrel' where owner = 'Diane';
35
36 -- 删除数据
37 DELETE FROM pet where name = 'squirrel';
38
39 -- 删除表
40 DROP TABLE myorder;
```

数据库大三大设计范式

- **1NF**

不可分割性，只要字段值还可以继续拆分，就不满足第一范式。

不可分割的意思就按字面理解就是最小单位，不能再分成更小单位了。

字段只能是一个值，不能被拆分成多个字段，否则的话，它就是可分割的，就不符合一范式。

- **2NF**

第二范式就是要有主键，要求其他字段都依赖于主键。

为什么要有主键？没有主键就没有唯一性，没有唯一性在集合中就定位不到这行记录，所以要主键。

在满足第一范式的前提下，主键外的每一列都必须完全依赖于主键。如果出现不完全依赖，只可能发生在联合主键的情况下。

- 3NF

在满足第二范式的前提下，除了主键列之外，其他列之间不能有传递依赖关系，即“消除冗余”。

消除冗余，就是各种信息只在一个地方存储，不出现在多张表中。

范式总结：范式，其实是用来学习参考的，设计的时候根据情况，未必一定要遵守，要灵活结合业务实际情况决定。

查询练习

- 准备数据

```
1  -- 创建数据库
2  CREATE DATABASE select_test;
3  -- 切换数据库
4  USE select_test;
5
6  -- 创建学生表
7  CREATE TABLE student (
8      no VARCHAR(20) PRIMARY KEY,
9      name VARCHAR(20) NOT NULL,
10     sex VARCHAR(10) NOT NULL,
11     birthday DATE, -- 生日
12     class VARCHAR(20) -- 所在班级
13 );
14
15 -- 创建教师表
16 CREATE TABLE teacher (
17     no VARCHAR(20) PRIMARY KEY,
18     name VARCHAR(20) NOT NULL,
19     sex VARCHAR(10) NOT NULL,
20     birthday DATE,
21     profession VARCHAR(20) NOT NULL, -- 职称
22     department VARCHAR(20) NOT NULL -- 部门
23 );
24
25 -- 创建课程表
```



```

26 CREATE TABLE course (
27     no VARCHAR(20) PRIMARY KEY,
28     name VARCHAR(20) NOT NULL,
29     t_no VARCHAR(20) NOT NULL, -- 教师编号
30     -- 表示该 tno 来自于 teacher 表中的 no 字段值
31     FOREIGN KEY(t_no) REFERENCES teacher(no)
32 );
33
34 -- 成绩表
35 CREATE TABLE score (
36     s_no VARCHAR(20) NOT NULL, -- 学生编号
37     c_no VARCHAR(20) NOT NULL, -- 课程号
38     degree DECIMAL, -- 成绩
39     -- 表示该 s_no, c_no 分别来自于 student, course 表中的 no 字段值
40     FOREIGN KEY(s_no) REFERENCES student(no),
41     FOREIGN KEY(c_no) REFERENCES course(no),
42     -- 设置 s_no, c_no 为联合主键
43     PRIMARY KEY(s_no, c_no)
44 );
45
46 -- 查看所有表
47 SHOW TABLES;
48
49 -- 添加学生表数据
50 INSERT INTO student VALUES('101', '曾华', '男', '1977-09-01', '95033');
51 INSERT INTO student VALUES('102', '匡明', '男', '1975-10-02', '95031');
52 INSERT INTO student VALUES('103', '王丽', '女', '1976-01-23', '95033');
53 INSERT INTO student VALUES('104', '李军', '男', '1976-02-20', '95033');
54 INSERT INTO student VALUES('105', '王芳', '女', '1975-02-10', '95031');
55 INSERT INTO student VALUES('106', '陆军', '男', '1974-06-03', '95031');
56 INSERT INTO student VALUES('107', '王尼玛', '男', '1976-02-20', '95033');
57 INSERT INTO student VALUES('108', '张全蛋', '男', '1975-02-10', '95031');
58 INSERT INTO student VALUES('109', '赵铁柱', '男', '1974-06-03', '95031');
59
60 -- 添加教师表数据
61 INSERT INTO teacher VALUES('804', '李诚', '男', '1958-12-02', '副教授', '计');
62 INSERT INTO teacher VALUES('856', '张旭', '男', '1969-03-12', '讲师', '电');
63 INSERT INTO teacher VALUES('825', '王萍', '女', '1972-05-05', '助教', '计');
64 INSERT INTO teacher VALUES('831', '刘冰', '女', '1977-08-14', '助教', '电');
65

```



```
66 -- 添加课程表数据
67 INSERT INTO course VALUES('3-105', '计算机导论', '825');
68 INSERT INTO course VALUES('3-245', '操作系统', '804');
69 INSERT INTO course VALUES('6-166', '数字电路', '856');
70 INSERT INTO course VALUES('9-888', '高等数学', '831');
71
72 -- 添加添加成绩表数据
73 INSERT INTO score VALUES('103', '3-105', '92');
74 INSERT INTO score VALUES('103', '3-245', '86');
75 INSERT INTO score VALUES('103', '6-166', '85');
76 INSERT INTO score VALUES('105', '3-105', '88');
77 INSERT INTO score VALUES('105', '3-245', '75');
78 INSERT INTO score VALUES('105', '6-166', '79');
79 INSERT INTO score VALUES('109', '3-105', '76');
80 INSERT INTO score VALUES('109', '3-245', '68');
81 INSERT INTO score VALUES('109', '6-166', '81');
82
83 -- 查看表结构
84 SELECT * FROM course;
85 SELECT * FROM score;
86 SELECT * FROM student;
87 SELECT * FROM teacher;
```

• 基础查询

```
1 -- 查询 student 表的所有行
2 SELECT * FROM student;
3
4 -- 查询 student 表中的 name、sex 和 class 字段的所有行
5 SELECT name, sex, class FROM student;
6
7 -- 查询 teacher 表中不重复的 department 列
8 -- department: 去重查询
9 SELECT DISTINCT department FROM teacher;
10
11 -- 查询 score 表中成绩在60-80之间的所有行（区间查询和运算符查询）
12 -- BETWEEN xx AND xx: 查询区间, AND 表示 "并且"
13 SELECT * FROM score WHERE degree BETWEEN 60 AND 80;
```



```

14 SELECT * FROM score WHERE degree > 60 AND degree < 80;
15
16 -- 查询 score 表中成绩为 85, 86 或 88 的行
17 -- IN: 查询规定中的多个值
18 SELECT * FROM score WHERE degree IN (85, 86, 88);
19
20 -- 查询 student 表中 '95031' 班或性别为 '女' 的所有行
21 -- or: 表示或者关系
22 SELECT * FROM student WHERE class = '95031' or sex = '女';
23
24 -- 以 class 降序的方式查询 student 表的所有行
25 -- DESC: 降序, 从高到低
26 -- ASC (默认): 升序, 从低到高
27 SELECT * FROM student ORDER BY class DESC;
28 SELECT * FROM student ORDER BY class ASC;
29
30 -- 以 c_no 升序、degree 降序查询 score 表的所有行
31 SELECT * FROM score ORDER BY c_no ASC, degree DESC;
32
33 -- 查询 "95031" 班的学生人数
34 -- COUNT: 统计
35 SELECT COUNT(*) FROM student WHERE class = '95031';
36
37 -- 查询 score 表中的最高分的学生学号和课程编号 (子查询或排序查询) 。
38 -- (SELECT MAX(degree) FROM score): 子查询, 算出最高分
39 SELECT s_no, c_no FROM score WHERE degree = (SELECT MAX(degree) FROM score);
40
41 -- 排序查询
42 -- LIMIT r, n: 表示从第r行开始, 查询n条数据
43 SELECT s_no, c_no, degree FROM score ORDER BY degree DESC LIMIT 0, 1;
44 -- LIMIT n offset r: 表示查询n条数据, 从第r行开始
45 SELECT s_no, c_no, degree FROM score ORDER BY degree DESC LIMIT 1 offset 1;

```

- 分组计算平均成绩

查询每门课的平均成绩

```

1 -- AVG: 平均值
2 SELECT AVG(degree) FROM score WHERE c_no = '3-105';

```



```

3 SELECT AVG(degree) FROM score WHERE c_no = '3-245';
4 SELECT AVG(degree) FROM score WHERE c_no = '6-166';
5
6 -- GROUP BY: 分组查询
7 SELECT c_no, AVG(degree) FROM score GROUP BY c_no;

```

• 分组条件与模糊查询

查询score表中至少有 2 名学生选修，并以 3 开头的课程的平均分数

分析表发现，至少有 2 名学生选修的课程是3-105、3-245、6-166，以 3 开头的课程是3-105、3-245。也就是说，我们要查询所有3-105和3-245的degree平均分。

```

1 -- 首先把 c_no, AVG(degree) 通过分组查询出来
2 SELECT c_no, AVG(degree) FROM score GROUP BY c_no
3 +-----+-----+
4 | c_no   | AVG(degree) |
5 +-----+-----+
6 | 3-105  |      85.3333 |
7 | 3-245  |      76.3333 |
8 | 6-166  |      81.6667 |
9 +-----+-----+
10
11 -- 再查询出至少有 2 名学生选修的课程
12 -- HAVING: 表示持有
13 HAVING COUNT(c_no) >= 2
14
15 -- 并且是以 3 开头的课程
16 -- LIKE 表示模糊查询, "%" 是一个通配符, 匹配 "3" 后面的任意字符。
17 AND c_no LIKE '3%';
18
19 -- 把前面的SQL语句拼接起来,
20 -- 后面加上一个 COUNT(*), 表示将每个分组的个数也查询出来。
21 SELECT c_no, AVG(degree), COUNT(*) FROM score GROUP BY c_no
22 HAVING COUNT(c_no) >= 2 AND c_no LIKE '3%';
23 +-----+-----+-----+
24 | c_no   | AVG(degree) | COUNT(*) |
25 +-----+-----+-----+
26 | 3-105  |      85.3333 |          3 |

```



```

27 | 3-245 | 76.3333 | 3 |
28 +-----+-----+-----+

```

• 多表查询 - 1

查询所有学生的name，以及该学生在score表中对应的c_no和degree

```

1  SELECT no, name FROM student;
2  +-----+-----+
3  | no   | name   |
4  +-----+-----+
5  | 101  | 曾华   |
6  | 102  | 匡明   |
7  | 103  | 王丽   |
8  | 104  | 李军   |
9  | 105  | 王芳   |
10 | 106  | 陆军   |
11 | 107  | 王尼玛 |
12 | 108  | 张全蛋 |
13 | 109  | 赵铁柱 |
14 +-----+-----+
15
16 SELECT s_no, c_no, degree FROM score;
17 +-----+-----+-----+
18 | s_no | c_no | degree |
19 +-----+-----+-----+
20 | 103  | 3-105 | 92     |
21 | 103  | 3-245 | 86     |
22 | 103  | 6-166 | 85     |
23 | 105  | 3-105 | 88     |
24 | 105  | 3-245 | 75     |
25 | 105  | 6-166 | 79     |
26 | 109  | 3-105 | 76     |
27 | 109  | 3-245 | 68     |
28 | 109  | 6-166 | 81     |
29 +-----+-----+-----+

```


通过分析可以发现，只要把score表中的s_no字段值替换成student表中对应的name字段值就可以了，如何做呢？

```

1  -- FROM...: 表示从 student, score 表中查询
2  -- WHERE 的条件表示为，只有在 student.no 和 score.s_no 相等时才显示出来。
3  SELECT name, c_no, degree FROM student, score
4  WHERE student.no = score.s_no;
5  +-----+-----+-----+
6  | name      | c_no  | degree |
7  +-----+-----+-----+
8  | 王丽      | 3-105 | 92     |
9  | 王丽      | 3-245 | 86     |
10 | 王丽      | 6-166 | 85     |
11 | 王芳      | 3-105 | 88     |
12 | 王芳      | 3-245 | 75     |
13 | 王芳      | 6-166 | 79     |
14 | 赵铁柱    | 3-105 | 76     |
15 | 赵铁柱    | 3-245 | 68     |
16 | 赵铁柱    | 6-166 | 81     |
17 +-----+-----+-----+

```

• 多表查询 - 2

查询所有学生的 no、课程名称 (course 表中的 name) 和成绩 (score 表中的 degree) 列。

只有score关联学生的no，因此只要查询score表，就能找出所有和学生相关的no和degree：

```

1  SELECT s_no, c_no, degree FROM score;
2  +-----+-----+-----+
3  | s_no | c_no  | degree |
4  +-----+-----+-----+
5  | 103  | 3-105 | 92     |
6  | 103  | 3-245 | 86     |
7  | 103  | 6-166 | 85     |
8  | 105  | 3-105 | 88     |
9  | 105  | 3-245 | 75     |
10 | 105  | 6-166 | 79     |
11 | 109  | 3-105 | 76     |

```



```

12 | 109 | 3-245 | 68 |
13 | 109 | 6-166 | 81 |
14 +-----+-----+-----+

```

然后查询course表：

```

1 SELECT no, name FROM course;
2 +-----+-----+
3 | no      | name                |
4 +-----+-----+
5 | 3-105   | 计算机导论          |
6 | 3-245   | 操作系统            |
7 | 6-166   | 数字电路            |
8 | 9-888   | 高等数学            |
9 +-----+-----+

```

只要把score表中的c_no替换成course表中对应的name字段值就可以了。

```

1 -- 增加一个查询字段 name，分别从 score、course 这两个表中查询。
2 -- as 表示取一个该字段的别名。
3 SELECT s_no, name as c_name, degree FROM score, course
4 WHERE score.c_no = course.no;
5 +-----+-----+-----+
6 | s_no | c_name                | degree |
7 +-----+-----+-----+
8 | 103  | 计算机导论            | 92     |
9 | 105  | 计算机导论            | 88     |
10 | 109  | 计算机导论            | 76     |
11 | 103  | 操作系统              | 86     |
12 | 105  | 操作系统              | 75     |
13 | 109  | 操作系统              | 68     |
14 | 103  | 数字电路              | 85     |
15 | 105  | 数字电路              | 79     |
16 | 109  | 数字电路              | 81     |
17 +-----+-----+-----+

```


- 三表关联查询

查询所有学生的 **name** 、课程名 (**course** 表中的 **name**) 和 **degree** 。

只有score表中关联学生的学号和课堂号，我们只要围绕着score这张表查询就好了。

```

1 SELECT * FROM score;
2 +-----+-----+-----+
3 | s_no | c_no | degree |
4 +-----+-----+-----+
5 | 103 | 3-105 | 92 |
6 | 103 | 3-245 | 86 |
7 | 103 | 6-166 | 85 |
8 | 105 | 3-105 | 88 |
9 | 105 | 3-245 | 75 |
10 | 105 | 6-166 | 79 |
11 | 109 | 3-105 | 76 |
12 | 109 | 3-245 | 68 |
13 | 109 | 6-166 | 81 |
14 +-----+-----+-----+

```

只要把 s_no 和 c_no 替换成 student 和 course 表中对应的 name 字段值就好了。

首先把 s_no 替换成 student 表中的 name 字段：

```

1 SELECT name, c_no, degree FROM student, score WHERE student.no = score.s
2 +-----+-----+-----+
3 | name      | c_no | degree |
4 +-----+-----+-----+
5 | 王丽      | 3-105 | 92 |
6 | 王丽      | 3-245 | 86 |
7 | 王丽      | 6-166 | 85 |
8 | 王芳      | 3-105 | 88 |
9 | 王芳      | 3-245 | 75 |
10 | 王芳      | 6-166 | 79 |
11 | 赵铁柱    | 3-105 | 76 |
12 | 赵铁柱    | 3-245 | 68 |
13 | 赵铁柱    | 6-166 | 81 |
14 +-----+-----+-----+

```


再把c_no替换成course表中的name字段：

```

1  -- 课程表
2  SELECT no, name FROM course;
3  +-----+-----+
4  | no      | name                |
5  +-----+-----+
6  | 3-105   | 计算机导论          |
7  | 3-245   | 操作系统            |
8  | 6-166   | 数字电路            |
9  | 9-888   | 高等数学            |
10 +-----+-----+
11
12 -- 由于字段名存在重复，使用 "表名.字段名 as 别名" 代替。
13 SELECT student.name as s_name, course.name as c_name, degree
14 FROM student, score, course
15 WHERE student.NO = score.s_no
16 AND score.c_no = course.no;

```

- 子查询加分组求平均分

查询95031班学生每门课程的平均成绩。

在score表中根据student表的学生编号筛选出学生的课堂号和成绩：

```

1  -- IN (..): 将筛选出的学生号当做 s_no 的条件查询
2  SELECT s_no, c_no, degree FROM score
3  WHERE s_no IN (SELECT no FROM student WHERE class = '95031');
4  +-----+-----+-----+
5  | s_no | c_no | degree |
6  +-----+-----+-----+
7  | 105  | 3-105 | 88     |
8  | 105  | 3-245 | 75     |
9  | 105  | 6-166 | 79     |
10 | 109  | 3-105 | 76     |
11 | 109  | 3-245 | 68     |

```



```

12 | 109 | 6-166 |      81 |
13 +-----+-----+-----+

```

这时只要将c_no分组一下就能得出95031班学生每门课的平均成绩：

```

1 SELECT c_no, AVG(degree) FROM score
2 WHERE s_no IN (SELECT no FROM student WHERE class = '95031')
3 GROUP BY c_no;
4 +-----+-----+
5 | c_no   | AVG(degree) |
6 +-----+-----+
7 | 3-105  |      82.0000 |
8 | 3-245  |      71.5000 |
9 | 6-166  |      80.0000 |
10 +-----+-----+

```

• 子查询 - 1

查询在 3-105 课程中，所有成绩高于 109 号同学的记录。

先用子查询查找出109同学在3-105中的成绩

```

1 select * from score
2 where c_no = '3-105' and s_no='109'

```

然后再以课程3-105为条件，查找成绩大76的记录

```

1 select * from score
2 where c_no = '3-105'
3 and degree>
4 (select degree from score
5 where c_no = '3-105' and s_no='109');

```

• 子查询 - 2

查询所有成绩高于 109 号同学的 3-105 课程成绩记录。

```
1  -- 不限制课程号，只要成绩大于109号同学的3-105课程成绩就可以。
2  SELECT * FROM score
3  WHERE degree > (SELECT degree FROM score WHERE s_no = '109' AND c_no = '3-105');
```

- YEAR 函数与带 IN 关键字查询

查询所有和 101、108 号学生同年出生的 no、name、birthday 列。

```
1  select no, name, birthday from student
2  where year(birthday) in (
3  select year(birthday) from student
4  where no in (101,108));
```

- 多层嵌套子查询

查询 '张旭' 教师任课的学生成绩表。

用的三张表teacher、course、score，首先找到教师编号：

```
1  SELECT NO FROM teacher WHERE NAME = '张旭'
```

通过course表找到该教师课程号：

```
1  select no from course
2  where t_no = (SELECT NO FROM teacher WHERE NAME = '张旭')
```

通过筛选出的课程号查询成绩表：

```
1  select * from score where c_no = (
2  select no from course where t_no = (
3  select no from teacher
```



```
4 where name = '张旭'));
```

- 多表查询

查询某选修课程多于5个同学的教师姓名。

首先在teacher表中，根据no字段来判断该教师的同一门课程是否有至少5名学员选修：

```
1 -- 查询 teacher 表
2 SELECT no, name FROM teacher;
3 +-----+-----+
4 | no   | name   |
5 +-----+-----+
6 | 804  | 李诚   |
7 | 825  | 王萍   |
8 | 831  | 刘冰   |
9 | 856  | 张旭   |
10 +-----+-----+
11
12 SELECT name FROM teacher WHERE no IN (
13     -- 在这里找到对应的条件
14 );
```

查看和教师编号有有关的表的信息：

```
1 SELECT * FROM course;
2 -- t_no: 教师编号
3 +-----+-----+-----+
4 | no   | name           | t_no |
5 +-----+-----+-----+
6 | 3-105 | 计算机导论     | 825  |
7 | 3-245 | 操作系统       | 804  |
8 | 6-166 | 数字电路       | 856  |
9 | 9-888 | 高等数学       | 831  |
10 +-----+-----+-----+
```


我们已经找到和教师编号有关的字段就在course表中，但是还无法知道哪门课程至少有5名学生选修，所以还需要根据score表来查询：

```

1  -- 在此之前向 score 插入一些数据，以便丰富查询条件。
2  INSERT INTO score VALUES ('101', '3-105', '90');
3  INSERT INTO score VALUES ('102', '3-105', '91');
4  INSERT INTO score VALUES ('104', '3-105', '89');
5
6  -- 查询 score 表
7  SELECT * FROM score;
8  +-----+-----+-----+
9  | s_no | c_no | degree |
10 +-----+-----+-----+
11 | 101  | 3-105 | 90     |
12 | 102  | 3-105 | 91     |
13 | 103  | 3-105 | 92     |
14 | 103  | 3-245 | 86     |
15 | 103  | 6-166 | 85     |
16 | 104  | 3-105 | 89     |
17 | 105  | 3-105 | 88     |
18 | 105  | 3-245 | 75     |
19 | 105  | 6-166 | 79     |
20 | 109  | 3-105 | 76     |
21 | 109  | 3-245 | 68     |
22 | 109  | 6-166 | 81     |
23 +-----+-----+-----+
24
25 -- 在 score 表中将 c_no 作为分组，并且限制 c_no 持有至少 5 条数据。
26 SELECT c_no FROM score GROUP BY c_no HAVING COUNT(*) > 5;
27 +-----+
28 | c_no |
29 +-----+
30 | 3-105 |
31 +-----+

```

根据筛选出来的课程号，找出在某课程中，拥有至少5名学员的教师编号：

```

1  SELECT t_no FROM course WHERE no IN (

```



```

2      SELECT c_no FROM score GROUP BY c_no HAVING COUNT(*) > 5
3 );
4 +-----+
5 | t_no |
6 +-----+
7 | 825 |
8 +-----+在teacher表中，根据筛选出来的教师编号找到教师姓名：SELECT name FROM teac
9      -- 最终条件
10     SELECT t_no FROM course WHERE no IN (
11         SELECT c_no FROM score GROUP BY c_no HAVING COUNT(*) > 5
12     )
13 );

```

• 子查询 - 3

查询“计算机系”课程的成绩表。

思路是，先找出teacher表中所有计算机系课程的编号，根据这个编号查询course表中的课程编号，再用课程编号查找score表

```

1  -- 通过 teacher 表查询所有 `计算机系` 的教师编号
2  SELECT no, name, department FROM teacher WHERE department = '计算机系'
3  +-----+-----+-----+
4  | no  | name  | department |
5  +-----+-----+-----+
6  | 804 | 李诚  | 计算机系   |
7  | 825 | 王萍  | 计算机系   |
8  +-----+-----+-----+
9
10 -- 通过 course 表查询该教师的课程编号
11 SELECT no FROM course WHERE t_no IN (
12     SELECT no FROM teacher WHERE department = '计算机系'
13 );
14 +-----+
15 | no  |
16 +-----+
17 | 3-245 |
18 | 3-105 |

```



```

19  +-----+
20
21  -- 根据筛选出来的课程号查询成绩表
22  SELECT * FROM score WHERE c_no IN (
23      SELECT no FROM course WHERE t_no IN (
24          SELECT no FROM teacher WHERE department = '计算机系'
25      )
26  );
27  +-----+-----+-----+
28  | s_no | c_no | degree |
29  +-----+-----+-----+
30  | 103  | 3-245 | 86 |
31  | 105  | 3-245 | 75 |
32  | 109  | 3-245 | 68 |
33  | 101  | 3-105 | 90 |
34  | 102  | 3-105 | 91 |
35  | 103  | 3-105 | 92 |
36  | 104  | 3-105 | 89 |
37  | 105  | 3-105 | 88 |
38  | 109  | 3-105 | 76 |
39  +-----+-----+-----+

```

• UNION 和 NOT IN 的使用

查询计算机系与电子工程系中的不同职称的教师。

```

1  +-----+-----+-----+-----+-----+-----+-----+
2  | no  | name | sex | birthday | profession | department |
3  +-----+-----+-----+-----+-----+-----+-----+
4  | 804 | 李诚 | 男  | 1958-12-02 | 副教授    | 计算机系  |
5  | 825 | 王萍 | 女  | 1972-05-05 | 助教      | 计算机系  |
6  | 831 | 刘冰 | 女  | 1977-08-14 | 助教      | 电子工程系 |
7  | 856 | 张旭 | 男  | 1969-03-12 | 讲师      | 电子工程系 |
8  +-----+-----+-----+-----+-----+-----+-----+
9
10 -- NOT: 代表逻辑非
11 SELECT * FROM teacher WHERE department = '计算机系' AND profession NOT IN
12     SELECT profession FROM teacher WHERE department = '电子工程系'

```



```

13 )
14 -- 合并两个集
15 UNION
16 SELECT * FROM teacher WHERE department = '电子工程系' AND profession NOT IN
17     SELECT profession FROM teacher WHERE department = '计算机系'
18 );

```

- **ANY** 表示至少一个 - **DESC** (降序)

查询课程 3-105 且成绩 至少 高于 3-245 的 **score** 表。

```

1  SELECT * FROM score WHERE c_no = '3-105';
2  +-----+-----+-----+
3  | s_no | c_no | degree |
4  +-----+-----+-----+
5  | 101  | 3-105 | 90 |
6  | 102  | 3-105 | 91 |
7  | 103  | 3-105 | 92 |
8  | 104  | 3-105 | 89 |
9  | 105  | 3-105 | 88 |
10 | 109  | 3-105 | 76 |
11 +-----+-----+-----+
12
13 SELECT * FROM score WHERE c_no = '3-245';
14 +-----+-----+-----+
15 | s_no | c_no | degree |
16 +-----+-----+-----+
17 | 103  | 3-245 | 86 |
18 | 105  | 3-245 | 75 |
19 | 109  | 3-245 | 68 |
20 +-----+-----+-----+
21
22 -- ANY: 符合SQL语句中的任意条件。
23 -- 也就是说，在 3-105 成绩中，只要有一个大于从 3-245 筛选出来的任意行就符合条件，
24 -- 最后根据降序查询结果。
25 SELECT * FROM score WHERE c_no = '3-105' AND degree > ANY(
26     SELECT degree FROM score WHERE c_no = '3-245'
27 ) ORDER BY degree DESC;

```



```

28 +-----+-----+-----+
29 | s_no | c_no | degree |
30 +-----+-----+-----+
31 | 103 | 3-105 | 92 |
32 | 102 | 3-105 | 91 |
33 | 101 | 3-105 | 90 |
34 | 104 | 3-105 | 89 |
35 | 105 | 3-105 | 88 |
36 | 109 | 3-105 | 76 |
37 +-----+-----+-----+

```

- 表示所有的 ALL

查询课程 3-105 且成绩高于 3-245 的 score 表。

```

1  -- 只需对上一道题稍作修改。
2  -- ALL：符合SQL语句中的所有条件。
3  -- 也就是说，在 3-105 每一行成绩中，都要大于从 3-245 筛选出来全部行才算符合条件。
4  SELECT * FROM score WHERE c_no = '3-105' AND degree > ALL(
5      SELECT degree FROM score WHERE c_no = '3-245'
6  );
7  +-----+-----+-----+
8  | s_no | c_no | degree |
9  +-----+-----+-----+
10 | 101 | 3-105 | 90 |
11 | 102 | 3-105 | 91 |
12 | 103 | 3-105 | 92 |
13 | 104 | 3-105 | 89 |
14 | 105 | 3-105 | 88 |
15 +-----+-----+-----+

```

- 复制表的数据作为条件查询

查询某课程成绩比该课程平均成绩低的score表。

```

1  -- 查询平均分
2  SELECT c_no, AVG(degree) FROM score GROUP BY c_no;
3  b表

```



```

4  +-----+-----+
5  | c_no  | AVG(degree) |
6  +-----+-----+
7  | 3-105 |      87.6667 |
8  | 3-245 |      76.3333 |
9  | 6-166 |      81.6667 |
10 +-----+-----+
11
12 -- 查询 score 表
13 select * from score;
14 a表
15 +-----+-----+-----+
16 | s_no | c_no  | degree |
17 +-----+-----+-----+
18 | 101  | 3-105 |      90 |
19 | 102  | 3-105 |      91 |
20 | 103  | 3-105 |      92 |
21 | 103  | 3-245 |      86 |
22 | 103  | 6-166 |      85 |
23 | 104  | 3-105 |      89 |
24 | 105  | 3-105 |      88 |
25 | 105  | 3-245 |      75 |
26 | 105  | 6-166 |      79 |
27 | 109  | 3-105 |      76 |
28 | 109  | 3-245 |      68 |
29 | 109  | 6-166 |      81 |
30 +-----+-----+-----+
31
32 -- 将表 b 作用于表 a 中查询数据
33 -- score a (b): 将表声明为 a (b),
34 -- 如此就能用 a.c_no = b.c_no 作为条件执行查询了。
35 SELECT * FROM score a WHERE degree < (
36     (SELECT AVG(degree) FROM score b WHERE a.c_no = b.c_no)
37 );
38 +-----+-----+-----+
39 | s_no | c_no  | degree |
40 +-----+-----+-----+
41 | 105  | 3-245 |      75 |
42 | 105  | 6-166 |      79 |
43 | 109  | 3-105 |      76 |

```



```

44 | 109 | 3-245 | 68 |
45 | 109 | 6-166 | 81 |
46 +-----+-----+-----+

```

• 子查询 - 4

查询所有任课（在 **course** 表里有课程）教师的 **name** 和 **department**。

```

1 SELECT name, department FROM teacher WHERE no IN (SELECT t_no FROM course)
2 +-----+-----+
3 | name      | department      |
4 +-----+-----+
5 | 李诚      | 计算机系        |
6 | 王萍      | 计算机系        |
7 | 刘冰      | 电子工程系      |
8 | 张旭      | 电子工程系      |
9 +-----+-----+

```

• 条件加组筛选

查询 **student** 表中至少有 2 名男生的 **class**。

```

1 -- 查看学生表信息
2 SELECT * FROM student;
3 +-----+-----+-----+-----+-----+
4 | no  | name    | sex | birthday   | class |
5 +-----+-----+-----+-----+-----+
6 | 101 | 曾华    | 男  | 1977-09-01 | 95033 |
7 | 102 | 匡明    | 男  | 1975-10-02 | 95031 |
8 | 103 | 王丽    | 女  | 1976-01-23 | 95033 |
9 | 104 | 李军    | 男  | 1976-02-20 | 95033 |
10 | 105 | 王芳    | 女  | 1975-02-10 | 95031 |
11 | 106 | 陆军    | 男  | 1974-06-03 | 95031 |
12 | 107 | 王尼玛  | 男  | 1976-02-20 | 95033 |
13 | 108 | 张全蛋  | 男  | 1975-02-10 | 95031 |
14 | 109 | 赵铁柱  | 男  | 1974-06-03 | 95031 |
15 | 110 | 张飞    | 男  | 1974-06-03 | 95038 |
16 +-----+-----+-----+-----+-----+
17 -- 只查询性别为男，然后按 class 分组，并限制 class 行大于 1。

```



```

18 select * from student where sex='男'
19 group by class
20 having count(class)>=2;
21 +-----+-----+-----+-----+-----+
22 | no   | name | sex | birthday   | class |
23 +-----+-----+-----+-----+-----+
24 | 101  | 曾华 | 男  | 1977-09-01 | 95033 |
25 | 102  | 匡明 | 男  | 1975-10-02 | 95031 |
26 +-----+-----+-----+-----+-----+

```

- **NOT LIKE 模糊查询取反**

查询 **student** 表中不姓 "王" 的同学记录。

```

1  -- NOT: 取反
2  -- LIKE: 模糊查询
3  mysql> SELECT * FROM student WHERE name NOT LIKE '王%';
4  +-----+-----+-----+-----+-----+
5  | no   | name      | sex | birthday   | class |
6  +-----+-----+-----+-----+-----+
7  | 101  | 曾华      | 男  | 1977-09-01 | 95033 |
8  | 102  | 匡明      | 男  | 1975-10-02 | 95031 |
9  | 104  | 李军      | 男  | 1976-02-20 | 95033 |
10 | 106  | 陆军      | 男  | 1974-06-03 | 95031 |
11 | 108  | 张全蛋    | 男  | 1975-02-10 | 95031 |
12 | 109  | 赵铁柱    | 男  | 1974-06-03 | 95031 |
13 | 110  | 张飞      | 男  | 1974-06-03 | 95038 |
14 +-----+-----+-----+-----+-----+

```

- **YEAR 与 NOW 函数**

查询 **student** 表中每个学生的姓名和年龄。

```

1 select name, year(now())-year(birthday) as age
2 from student;
3 +-----+-----+
4 | name      | age |
5 +-----+-----+
6 | 曾华      | 42  |

```


7		匡明		44	
8		王丽		43	
9		李军		43	
10		王芳		44	
11		陆军		45	
12		王尼玛		43	
13		张全蛋		44	
14		赵铁柱		45	
15		张飞		45	
16	+-----+-----+				

• MAX 与 MIN 函数

查询 **student** 表中最大和最小的 **birthday** 值。

```

1 SELECT MAX(birthday), MIN(birthday) FROM student;
2 +-----+-----+
3 | MAX(birthday) | MIN(birthday) |
4 +-----+-----+
5 | 1977-09-01    | 1974-06-03    |
6 +-----+-----+

```

• 多段排序

以 **class** 和 **birthday** 从大到小的顺序查询 **student** 表。

```

1 SELECT * FROM student ORDER BY class DESC, birthday DESC;
2 +-----+-----+-----+-----+-----+
3 | no  | name  | sex | birthday  | class |
4 +-----+-----+-----+-----+-----+
5 | 110 | 张飞  | 男  | 1974-06-03 | 95038 |
6 | 101 | 曾华  | 男  | 1977-09-01 | 95033 |
7 | 104 | 李军  | 男  | 1976-02-20 | 95033 |
8 | 107 | 王尼玛 | 男  | 1976-02-20 | 95033 |
9 | 103 | 王丽  | 女  | 1976-01-23 | 95033 |
10 | 102 | 匡明  | 男  | 1975-10-02 | 95031 |
11 | 105 | 王芳  | 女  | 1975-02-10 | 95031 |
12 | 108 | 张全蛋 | 男  | 1975-02-10 | 95031 |
13 | 106 | 陆军  | 男  | 1974-06-03 | 95031 |

```



```

14 | 109 | 赵铁柱 | 男 | 1974-06-03 | 95031 |
15 +-----+-----+-----+-----+-----+

```

• 子查询 - 5

查询 "男" 教师及其所上的课程。

```

1 SELECT * FROM course WHERE t_no in (SELECT no FROM teacher WHERE sex = '男');
2 +-----+-----+-----+
3 | no      | name          | t_no |
4 +-----+-----+-----+
5 | 3-245   | 操作系统      | 804   |
6 | 6-166   | 数字电路      | 856   |
7 +-----+-----+-----+

```

• MAX 函数与子查询

查询最高分同学的 **score** 表。

```

1 -- 找出最高成绩 (该查询只能有一个结果)
2 SELECT MAX(degree) FROM score;
3
4 -- 根据上面的条件筛选出所有最高成绩表,
5 -- 该查询可能有多个结果, 假设 degree 值多次符合条件。
6 SELECT * FROM score WHERE degree = (SELECT MAX(degree) FROM score);
7 +-----+-----+-----+
8 | s_no | c_no | degree |
9 +-----+-----+-----+
10 | 103  | 3-105 | 92      |
11 +-----+-----+-----+

```

• 子查询 - 6

查询和 "李军" 同性别的所有同学 **name** 。

```

1 select name from student where sex = (
2 SELECT sex FROM student where name='李军'
3 );

```



```

4  +-----+
5  |  name  |
6  +-----+
7  |  曾华  |
8  |  匡明  |
9  |  李军  |
10 |  陆军  |
11 |  王尼玛 |
12 |  张全蛋 |
13 |  赵铁柱 |
14 |  张飞  |
15 +-----+

```

• 子查询 - 7

查询和 "李军" 同性别且同班的同学 **name** 。

```

1  SELECT name, sex, class FROM student WHERE sex = (
2      SELECT sex FROM student WHERE name = '李军'
3  ) AND class = (
4      SELECT class FROM student WHERE name = '李军'
5  );
6  +-----+-----+-----+
7  |  name      | sex | class |
8  +-----+-----+-----+
9  |  曾华      | 男  | 95033 |
10 |  李军      | 男  | 95033 |
11 |  王尼玛    | 男  | 95033 |
12 +-----+-----+-----+

```

• 子查询 - 8

查询所有选修 "计算机导论" 课程的 "男" 同学成绩表。

需要的 "计算机导论" 和性别为 "男" 的编号可以在 `course` 和 `student` 表中找到。

```

1  SELECT * FROM score WHERE c_no = (
2      SELECT no FROM course WHERE name = '计算机导论'
3  ) AND s_no IN (

```



```

4      SELECT no FROM student WHERE sex = '男'
5 );
6 +-----+-----+-----+
7 | s_no | c_no | degree |
8 +-----+-----+-----+
9 | 101  | 3-105 | 90 |
10 | 102  | 3-105 | 91 |
11 | 104  | 3-105 | 89 |
12 | 109  | 3-105 | 76 |
13 +-----+-----+-----+

```

- 按等级查询

建立一个 grade 表代表学生的成绩等级，并插入数据：

```

1 CREATE TABLE grade (
2     low INT(3),
3     upp INT(3),
4     grade char(1)
5 );
6
7 INSERT INTO grade VALUES (90, 100, 'A');
8 INSERT INTO grade VALUES (80, 89, 'B');
9 INSERT INTO grade VALUES (70, 79, 'C');
10 INSERT INTO grade VALUES (60, 69, 'D');
11 INSERT INTO grade VALUES (0, 59, 'E');
12
13 SELECT * FROM grade;
14 +-----+-----+-----+
15 | low | upp | grade |
16 +-----+-----+-----+
17 | 90 | 100 | A |
18 | 80 | 89 | B |
19 | 70 | 79 | C |
20 | 60 | 69 | D |
21 | 0 | 59 | E |
22 +-----+-----+-----+

```


- 查询所有学生的 `s_no`、`c_no` 和 `grade` 列。

思路是，使用区间（`BETWEEN`）查询，判断学生的成绩（`degree`）在 `grade` 表的 `low` 和 `upp` 之间。

```

1 SELECT s_no, c_no, grade FROM score, grade
2 WHERE degree BETWEEN low AND upp;
3 +-----+-----+-----+
4 | s_no | c_no | grade |
5 +-----+-----+-----+
6 | 101 | 3-105 | A      |
7 | 102 | 3-105 | A      |
8 | 103 | 3-105 | A      |
9 | 103 | 3-245 | B      |
10 | 103 | 6-166 | B      |
11 | 104 | 3-105 | B      |
12 | 105 | 3-105 | B      |
13 | 105 | 3-245 | C      |
14 | 105 | 6-166 | C      |
15 | 109 | 3-105 | C      |
16 | 109 | 3-245 | D      |
17 | 109 | 6-166 | B      |
18 +-----+-----+-----+

```

- 连接查询

准备用于测试连接查询的数据：

```

1 CREATE DATABASE testJoin;
2
3 CREATE TABLE person (
4     id INT,
5     name VARCHAR(20),
6     cardId INT
7 );
8
9 CREATE TABLE card (
10     id INT,
11     name VARCHAR(20)

```



```

12 );
13
14 INSERT INTO card VALUES (1, '饭卡'), (2, '建行卡'), (3, '农行卡'), (4, '工商
15 SELECT * FROM card;
16 +-----+-----+
17 | id    | name      |
18 +-----+-----+
19 |     1 | 饭卡      |
20 |     2 | 建行卡    |
21 |     3 | 农行卡    |
22 |     4 | 工商卡    |
23 |     5 | 邮政卡    |
24 +-----+-----+
25
26 INSERT INTO person VALUES (1, '张三', 1), (2, '李四', 3), (3, '王五', 6);
27 SELECT * FROM person;
28 +-----+-----+-----+
29 | id    | name      | cardId |
30 +-----+-----+-----+
31 |     1 | 张三      |     1  |
32 |     2 | 李四      |     3  |
33 |     3 | 王五      |     6  |
34 +-----+-----+-----+

```

分析两张表发现，person 表并没有为 cardId 字段设置一个在 card 表中对应的 id 外键。如果设置的话，person 中 cardId 字段值为 6 的行就插不进去，因为该 cardId 值在 card 表中并没有。

• 内连接

要查询这两张表中有关系的数据，可以使用 INNER JOIN (内连接) 将它们连接在一起。

```

1  -- INNER JOIN: 表示为内连接，将两张表拼接在一起。
2  -- on: 表示要执行某个条件。
3  SELECT * FROM person INNER JOIN card on person.cardId = card.id;
4  +-----+-----+-----+-----+-----+
5  | id    | name      | cardId | id    | name      |
6  +-----+-----+-----+-----+-----+

```



```

7 |      1 | 张三    |      1 |      1 | 饭卡      |
8 |      2 | 李四    |      3 |      3 | 农行卡    |
9 +-----+-----+-----+-----+-----+
10
11 -- 将 INNER 关键字省略掉，结果也是一样的。
12 -- SELECT * FROM person JOIN card on person.cardId = card.id;注意: card 的

```

• 左外连接

完整显示左边的表 (person)，右边的表如果符合条件就显示，不符合则补 NULL 。

```

1 -- LEFT JOIN 也叫做 LEFT OUTER JOIN，用这两种方式的查询结果是一样的。
2 SELECT * FROM person LEFT JOIN card on person.cardId = card.id;
3 +-----+-----+-----+-----+-----+
4 | id    | name   | cardId | id    | name   |
5 +-----+-----+-----+-----+-----+
6 |      1 | 张三   |      1 | 1     | 饭卡   |
7 |      2 | 李四   |      3 | 3     | 农行卡 |
8 |      3 | 王五   |      6 | NULL  | NULL   |
9 +-----+-----+-----+-----+-----+

```

• 右外链接

完整显示右边的表 (card)，左边的表如果符合条件就显示，不符合则补 NULL 。

```

1 SELECT * FROM person RIGHT JOIN card on person.cardId = card.id;
2 +-----+-----+-----+-----+-----+
3 | id    | name   | cardId | id    | name   |
4 +-----+-----+-----+-----+-----+
5 |      1 | 张三   |      1 | 1     | 饭卡   |
6 |      2 | 李四   |      3 | 3     | 农行卡 |
7 | NULL  | NULL   | NULL  | 2     | 建行卡 |
8 | NULL  | NULL   | NULL  | 4     | 工商卡 |
9 | NULL  | NULL   | NULL  | 5     | 邮政卡 |
10 +-----+-----+-----+-----+-----+

```

• 全外链接

完整显示两张表的全部数据。

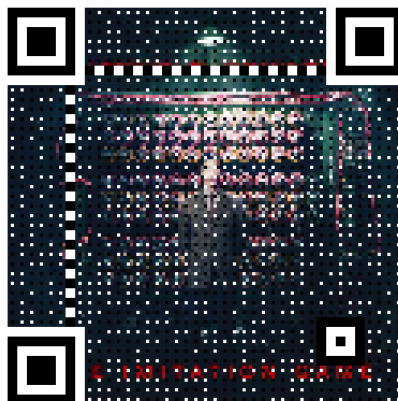
```

1  -- MySQL 不支持这种语法的全外连接
2  -- SELECT * FROM person FULL JOIN card on person.cardId = card.id;
3  -- 出现错误:
4  -- ERROR 1054 (42S22): Unknown column 'person.cardId' in 'on clause'
5
6  -- MySQL全连接语法, 使用 UNION 将两张表合并在一起。
7  SELECT * FROM person LEFT JOIN card on person.cardId = card.id
8  UNION
9  SELECT * FROM person RIGHT JOIN card on person.cardId = card.id;
10 +-----+-----+-----+-----+-----+
11 | id   | name   | cardId | id   | name   |
12 +-----+-----+-----+-----+-----+
13 | 1    | 张三   | 1       | 1    | 饭卡   |
14 | 2    | 李四   | 3       | 3    | 农行卡 |
15 | 3    | 王五   | 6       | NULL | NULL   |
16 | NULL | NULL   | NULL    | 2    | 建行卡 |
17 | NULL | NULL   | NULL    | 4    | 工商卡 |
18 | NULL | NULL   | NULL    | 5    | 邮政卡 |
19 +-----+-----+-----+-----+-----+

```

全文到这里撒花完结~

注：文章首发于知乎专栏，已授权数据不吹牛原创发表。



欢迎大家关注远辰同学知乎专栏！

推荐阅读



高级分析师是怎么玩转Excel的？



我用Python发现了隐藏的6大秘密..



数据分析里常用的五个统计学概念

『数据不吹牛』二群限额开启
后台回复“入群”
即可加入

“干货！”🌟