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

```
-- 显示所有数据库
show databases;

-- 创建数据库
CREATE DATABASE test;

-- 切换数据库
use test;

-- 显示数据库中的所有表
show tables;

-- 创建数据表
CREATE TABLE pet (
    name VARCHAR(20),
    owner VARCHAR(20),
    species VARCHAR(20),
    sex CHAR(1),
    birth DATE,
    death DATE
);

-- 查看数据表结构
```

```
-- describe pet;
desc pet;

-- 查询表
SELECT * from pet;

-- 插入数据
INSERT INTO pet VALUES ('puffball', 'Diane', 'hamster', 'f', '1990-03-3(

-- 修改数据
UPDATE pet SET name = 'squirrel' where owner = 'Diane';

-- 删除数据
DELETE FROM pet where name = 'squirrel';

-- 删除表
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**

```sql
-- 创建数据库
CREATE DATABASE select_test;
-- 切换数据库
USE select_test;

-- 创建学生表
CREATE TABLE student (
    no VARCHAR(20) PRIMARY KEY,
    name VARCHAR(20) NOT NULL,
    sex VARCHAR(10) NOT NULL,
    birthday DATE, -- 生日
    class VARCHAR(20) -- 所在班级
);

-- 创建教师表
CREATE TABLE teacher (
    no VARCHAR(20) PRIMARY KEY,
    name VARCHAR(20) NOT NULL,
    sex VARCHAR(10) NOT NULL,
    birthday DATE,
    profession VARCHAR(20) NOT NULL, -- 职称
    department VARCHAR(20) NOT NULL -- 部门
);

-- 创建课程表
```

```
CREATE TABLE course (
    no VARCHAR(20) PRIMARY KEY,
    name VARCHAR(20) NOT NULL,
    t_no VARCHAR(20) NOT NULL, -- 教师编号
    -- 表示该 tno 来自于 teacher 表中的 no 字段值
    FOREIGN KEY(t_no) REFERENCES teacher(no)
);

-- 成绩表
CREATE TABLE score (
    s_no VARCHAR(20) NOT NULL, -- 学生编号
    c_no VARCHAR(20) NOT NULL, -- 课程号
    degree DECIMAL,  -- 成绩
    -- 表示该 s_no, c_no 分别来自于 student, course 表中的 no 字段值
    FOREIGN KEY(s_no) REFERENCES student(no),
    FOREIGN KEY(c_no) REFERENCES course(no),
    -- 设置 s_no, c_no 为联合主键
    PRIMARY KEY(s_no, c_no)
);

-- 查看所有表
SHOW TABLES;

-- 添加学生表数据
INSERT INTO student VALUES('101', '曾华', '男', '1977-09-01', '95033');
INSERT INTO student VALUES('102', '匡明', '男', '1975-10-02', '95031');
INSERT INTO student VALUES('103', '王丽', '女', '1976-01-23', '95033');
INSERT INTO student VALUES('104', '李军', '男', '1976-02-20', '95033');
INSERT INTO student VALUES('105', '王芳', '女', '1975-02-10', '95031');
INSERT INTO student VALUES('106', '陆军', '男', '1974-06-03', '95031');
INSERT INTO student VALUES('107', '王尼玛', '男', '1976-02-20', '95033');
INSERT INTO student VALUES('108', '张全蛋', '男', '1975-02-10', '95031');
INSERT INTO student VALUES('109', '赵铁柱', '男', '1974-06-03', '95031');

-- 添加教师表数据
INSERT INTO teacher VALUES('804', '李诚', '男', '1958-12-02', '副教授', '计
INSERT INTO teacher VALUES('856', '张旭', '男', '1969-03-12', '讲师', '电子
INSERT INTO teacher VALUES('825', '王萍', '女', '1972-05-05', '助教', '计算
INSERT INTO teacher VALUES('831', '刘冰', '女', '1977-08-14', '助教', '电子

```

```sql
-- 添加课程表数据
INSERT INTO course VALUES('3-105', '计算机导论', '825');
INSERT INTO course VALUES('3-245', '操作系统', '804');
INSERT INTO course VALUES('6-166', '数字电路', '856');
INSERT INTO course VALUES('9-888', '高等数学', '831');

-- 添加添加成绩表数据
INSERT INTO score VALUES('103', '3-105', '92');
INSERT INTO score VALUES('103', '3-245', '86');
INSERT INTO score VALUES('103', '6-166', '85');
INSERT INTO score VALUES('105', '3-105', '88');
INSERT INTO score VALUES('105', '3-245', '75');
INSERT INTO score VALUES('105', '6-166', '79');
INSERT INTO score VALUES('109', '3-105', '76');
INSERT INTO score VALUES('109', '3-245', '68');
INSERT INTO score VALUES('109', '6-166', '81');

-- 查看表结构
SELECT * FROM course;
SELECT * FROM score;
SELECT * FROM student;
SELECT * FROM teacher;
```

- **基础查询**

```sql
-- 查询 student 表的所有行
SELECT * FROM student;

-- 查询 student 表中的 name、sex 和 class 字段的所有行
SELECT name, sex, class FROM student;

-- 查询 teacher 表中不重复的 department 列
-- department: 去重查询
SELECT DISTINCT department FROM teacher;

-- 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）
-- BETWEEN xx AND xx: 查询区间, AND 表示 "并且"
SELECT * FROM score WHERE degree BETWEEN 60 AND 80;
```

```sql
SELECT * FROM score WHERE degree > 60 AND degree < 80;

-- 查询 score 表中成绩为 85, 86 或 88 的行
-- IN: 查询规定中的多个值
SELECT * FROM score WHERE degree IN (85, 86, 88);

-- 查询 student 表中 '95031' 班或性别为 '女' 的所有行
-- or: 表示或者关系
SELECT * FROM student WHERE class = '95031' or sex = '女';

-- 以 class 降序的方式查询 student 表的所有行
-- DESC: 降序，从高到低
-- ASC（默认）: 升序，从低到高
SELECT * FROM student ORDER BY class DESC;
SELECT * FROM student ORDER BY class ASC;

-- 以 c_no 升序、degree 降序查询 score 表的所有行
SELECT * FROM score ORDER BY c_no ASC, degree DESC;

-- 查询 "95031" 班的学生人数
-- COUNT: 统计
SELECT COUNT(*) FROM student WHERE class = '95031';

-- 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）。
-- (SELECT MAX(degree) FROM score): 子查询，算出最高分
SELECT s_no, c_no FROM score WHERE degree = (SELECT MAX(degree) FROM sc

--  排序查询
-- LIMIT r, n: 表示从第r行开始，查询n条数据
SELECT s_no, c_no, degree FROM score ORDER BY degree DESC LIMIT 0, 1;
-- LIMIT n offset r: 表示查询n条数据,从第r行开始
SELECT s_no, c_no, degree FROM score ORDER BY degree DESC LIMIT 1 offse
```

- **分组计算平均成绩**

查询每门课的平均成绩

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

```
SELECT AVG(degree) FROM score WHERE c_no = '3-245';
SELECT AVG(degree) FROM score WHERE c_no = '6-166';

-- GROUP BY: 分组查询
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。

```
-- 首先把 c_no, AVG(degree) 通过分组查询出来
SELECT c_no, AVG(degree) FROM score GROUP BY c_no
+-------+-------------+
| c_no  | AVG(degree) |
+-------+-------------+
| 3-105 |     85.3333 |
| 3-245 |     76.3333 |
| 6-166 |     81.6667 |
+-------+-------------+

-- 再查询出至少有 2 名学生选修的课程
-- HAVING: 表示持有
HAVING COUNT(c_no) >= 2

-- 并且是以 3 开头的课程
-- LIKE 表示模糊查询，"%" 是一个通配符，匹配 "3" 后面的任意字符。
AND c_no LIKE '3%';

-- 把前面的SQL语句拼接起来，
-- 后面加上一个 COUNT(*)，表示将每个分组的个数也查询出来。
SELECT c_no, AVG(degree), COUNT(*) FROM score GROUP BY c_no
HAVING COUNT(c_no) >= 2 AND c_no LIKE '3%';
+-------+-------------+----------+
| c_no  | AVG(degree) | COUNT(*) |
+-------+-------------+----------+
| 3-105 |     85.3333 |        3 |
```

```
| 3-245 |     76.3333 |         3 |
+-------+--------------+-----------+
```

- **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

```
SELECT no, name FROM student;
+-----+-----------+
| no  | name      |
+-----+-----------+
| 101 | 曾华      |
| 102 | 匡明      |
| 103 | 王丽      |
| 104 | 李军      |
| 105 | 王芳      |
| 106 | 陆军      |
| 107 | 王尼玛    |
| 108 | 张全蛋    |
| 109 | 赵铁柱    |
+-----+-----------+

SELECT s_no, c_no, degree FROM score;
+------+-------+--------+
| s_no | c_no  | degree |
+------+-------+--------+
| 103  | 3-105 |     92 |
| 103  | 3-245 |     86 |
| 103  | 6-166 |     85 |
| 105  | 3-105 |     88 |
| 105  | 3-245 |     75 |
| 105  | 6-166 |     79 |
| 109  | 3-105 |     76 |
| 109  | 3-245 |     68 |
| 109  | 6-166 |     81 |
+------+-------+--------+
```

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?

```
-- FROM...: 表示从 student, score 表中查询
-- WHERE 的条件表示为，只有在 student.no 和 score.s_no 相等时才显示出来。
SELECT name, c_no, degree FROM student, score
WHERE student.no = score.s_no;
+-----------+-------+--------+
| name      | c_no  | degree |
+-----------+-------+--------+
| 王丽      | 3-105 |     92 |
| 王丽      | 3-245 |     86 |
| 王丽      | 6-166 |     85 |
| 王芳      | 3-105 |     88 |
| 王芳      | 3-245 |     75 |
| 王芳      | 6-166 |     79 |
| 赵铁柱    | 3-105 |     76 |
| 赵铁柱    | 3-245 |     68 |
| 赵铁柱    | 6-166 |     81 |
+-----------+-------+--------+
```

- **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：

```
SELECT s_no, c_no, degree FROM score;
+------+-------+--------+
| s_no | c_no  | degree |
+------+-------+--------+
| 103  | 3-105 |     92 |
| 103  | 3-245 |     86 |
| 103  | 6-166 |     85 |
| 105  | 3-105 |     88 |
| 105  | 3-245 |     75 |
| 105  | 6-166 |     79 |
| 109  | 3-105 |     76 |
```

```
| 109  | 3-245 |     68 |
| 109  | 6-166 |     81 |
+------+-------+--------+
```

然后查询course表：

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

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。

```
-- 增加一个查询字段 name，分别从 score、course 这两个表中查询。
-- as 表示取一个该字段的别名。
SELECT s_no, name as c_name, degree FROM score, course
WHERE score.c_no = course.no;
+------+-----------------+--------+
| s_no | c_name          | degree |
+------+-----------------+--------+
| 103  | 计算机导论      |     92 |
| 105  | 计算机导论      |     88 |
| 109  | 计算机导论      |     76 |
| 103  | 操作系统        |     86 |
| 105  | 操作系统        |     75 |
| 109  | 操作系统        |     68 |
| 103  | 数字电路        |     85 |
| 105  | 数字电路        |     79 |
| 109  | 数字电路        |     81 |
+------+-----------------+--------+
```

- **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。

```
SELECT * FROM score;
+------+-------+--------+
| s_no | c_no  | degree |
+------+-------+--------+
| 103  | 3-105 |     92 |
| 103  | 3-245 |     86 |
| 103  | 6-166 |     85 |
| 105  | 3-105 |     88 |
| 105  | 3-245 |     75 |
| 105  | 6-166 |     79 |
| 109  | 3-105 |     76 |
| 109  | 3-245 |     68 |
| 109  | 6-166 |     81 |
+------+-------+--------+
```

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：

```
SELECT name, c_no, degree FROM student, score WHERE student.no = score.s
+-----------+-------+--------+
| name      | c_no  | degree |
+-----------+-------+--------+
| 王丽      | 3-105 |     92 |
| 王丽      | 3-245 |     86 |
| 王丽      | 6-166 |     85 |
| 王芳      | 3-105 |     88 |
| 王芳      | 3-245 |     75 |
| 王芳      | 6-166 |     79 |
| 赵铁柱    | 3-105 |     76 |
| 赵铁柱    | 3-245 |     68 |
| 赵铁柱    | 6-166 |     81 |
+-----------+-------+--------+
```

再把c_no替换成course表中的name字段：

```
-- 课程表
SELECT no, name FROM course;
+-------+-----------------+
| no    | name            |
+-------+-----------------+
| 3-105 | 计算机导论      |
| 3-245 | 操作系统        |
| 6-166 | 数字电路        |
| 9-888 | 高等数学        |
+-------+-----------------+

-- 由于字段名存在重复，使用 "表名.字段名 as 别名" 代替。
SELECT student.name as s_name, course.name as c_name, degree
FROM student, score, course
WHERE student.NO = score.s_no
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：

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

```
| 109  | 6-166 |     81 |
+------+-------+--------+
```

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：

```
SELECT c_no, AVG(degree) FROM score
WHERE s_no IN (SELECT no FROM student WHERE class = '95031')
GROUP BY c_no;
+-------+-------------+
| c_no  | AVG(degree) |
+-------+-------------+
| 3-105 |     82.0000 |
| 3-245 |     71.5000 |
| 6-166 |     80.0000 |
+-------+-------------+
```

- **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

```
select * from score
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

```
select * from score
where c_no = '3-105'
and degree>
(select degree from score
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。

```
-- 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。
SELECT * FROM score
WHERE degree > (SELECT degree FROM score WHERE s_no = '109' AND c_no = '3
```

- **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。

```
select no, name, birthday from student
where year(birthday) in (
select year(birthday) from student
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：

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

通过sourse表找到该教师课程号：

```
select no from course
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：

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

```
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：

```
-- 查询 teacher 表
SELECT no, name FROM teacher;
+-----+--------+
| no  | name   |
+-----+--------+
| 804 | 李诚   |
| 825 | 王萍   |
| 831 | 刘冰   |
| 856 | 张旭   |
+-----+--------+

SELECT name FROM teacher WHERE no IN (
    -- 在这里找到对应的条件
);
```

查看和教师编号有有关的表的信息：

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

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：

```
-- 在此之前向 score 插入一些数据，以便丰富查询条件。
INSERT INTO score VALUES ('101', '3-105', '90');
INSERT INTO score VALUES ('102', '3-105', '91');
INSERT INTO score VALUES ('104', '3-105', '89');

-- 查询 score 表
SELECT * FROM score;
+------+-------+--------+
| s_no | c_no  | degree |
+------+-------+--------+
| 101  | 3-105 |     90 |
| 102  | 3-105 |     91 |
| 103  | 3-105 |     92 |
| 103  | 3-245 |     86 |
| 103  | 6-166 |     85 |
| 104  | 3-105 |     89 |
| 105  | 3-105 |     88 |
| 105  | 3-245 |     75 |
| 105  | 6-166 |     79 |
| 109  | 3-105 |     76 |
| 109  | 3-245 |     68 |
| 109  | 6-166 |     81 |
+------+-------+--------+

-- 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。
SELECT c_no FROM score GROUP BY c_no HAVING COUNT(*) > 5;
+-------+
| c_no  |
+-------+
| 3-105 |
+-------+
```

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：

```
SELECT t_no FROM course WHERE no IN (
```

```
    SELECT c_no FROM score GROUP BY c_no HAVING COUNT(*) > 5
);
+------+
| t_no |
+------+
| 825  |
+------+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
    -- 最终条件
    SELECT t_no FROM course WHERE no IN (
        SELECT c_no FROM score GROUP BY c_no HAVING COUNT(*) > 5
    )
);
```

- **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

```
-- 通过 teacher 表查询所有 `计算机系` 的教师编号
SELECT no, name, department FROM teacher WHERE department = '计算机系'
+-----+--------+--------------+
| no  | name   | department   |
+-----+--------+--------------+
| 804 | 李诚   | 计算机系     |
| 825 | 王萍   | 计算机系     |
+-----+--------+--------------+

-- 通过 course 表查询该教师的课程编号
SELECT no FROM course WHERE t_no IN (
    SELECT no FROM teacher WHERE department = '计算机系'
);
+-------+
| no    |
+-------+
| 3-245 |
| 3-105 |
```

```
+-------+

-- 根据筛选出来的课程号查询成绩表
SELECT * FROM score WHERE c_no IN (
    SELECT no FROM course WHERE t_no IN (
        SELECT no FROM teacher WHERE department = '计算机系'
    )
);
+------+-------+--------+
| s_no | c_no  | degree |
+------+-------+--------+
| 103  | 3-245 |     86 |
| 105  | 3-245 |     75 |
| 109  | 3-245 |     68 |
| 101  | 3-105 |     90 |
| 102  | 3-105 |     91 |
| 103  | 3-105 |     92 |
| 104  | 3-105 |     89 |
| 105  | 3-105 |     88 |
| 109  | 3-105 |     76 |
+------+-------+--------+
```

- **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。

```
+-----+------+-----+-----------+-----------+------------+
| no  | name | sex | birthday  | profession | department |
+-----+------+-----+-----------+-----------+------------+
| 804 | 李诚 | 男  | 1958-12-02 | 副教授     | 计算机系   |
| 825 | 王萍 | 女  | 1972-05-05 | 助教       | 计算机系   |
| 831 | 刘冰 | 女  | 1977-08-14 | 助教       | 电子工程系 |
| 856 | 张旭 | 男  | 1969-03-12 | 讲师       | 电子工程系 |
+-----+------+-----+-----------+-----------+------------+

-- NOT: 代表逻辑非
SELECT * FROM teacher WHERE department = '计算机系' AND profession NOT IN
    SELECT profession FROM teacher WHERE department = '电子工程系'
```

```
)
-- 合并两个集
UNION
SELECT * FROM teacher WHERE department = '电子工程系' AND profession NOT I
    SELECT profession FROM teacher WHERE department = '计算机系'
);
```

- **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。

```
SELECT * FROM score WHERE c_no = '3-105';
+------+-------+--------+
| s_no | c_no  | degree |
+------+-------+--------+
| 101  | 3-105 |     90 |
| 102  | 3-105 |     91 |
| 103  | 3-105 |     92 |
| 104  | 3-105 |     89 |
| 105  | 3-105 |     88 |
| 109  | 3-105 |     76 |
+------+-------+--------+

SELECT * FROM score WHERE c_no = '3-245';
+------+-------+--------+
| s_no | c_no  | degree |
+------+-------+--------+
| 103  | 3-245 |     86 |
| 105  | 3-245 |     75 |
| 109  | 3-245 |     68 |
+------+-------+--------+

-- ANY: 符合SQL语句中的任意条件。
-- 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，
-- 最后根据降序查询结果。
SELECT * FROM score WHERE c_no = '3-105' AND degree > ANY(
    SELECT degree FROM score WHERE c_no = '3-245'
) ORDER BY degree DESC;
```

```
+------+-------+--------+
| s_no | c_no  | degree |
+------+-------+--------+
| 103  | 3-105 |     92 |
| 102  | 3-105 |     91 |
| 101  | 3-105 |     90 |
| 104  | 3-105 |     89 |
| 105  | 3-105 |     88 |
| 109  | 3-105 |     76 |
+------+-------+--------+
```

- **表示所有的 ALL**

查询课程 3-105 且成绩高于 3-245 的 score 表。

```
-- 只需对上一道题稍作修改。
-- ALL: 符合SQL语句中的所有条件。
-- 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。
SELECT * FROM score WHERE c_no = '3-105' AND degree > ALL(
    SELECT degree FROM score WHERE c_no = '3-245'
);
+------+-------+--------+
| s_no | c_no  | degree |
+------+-------+--------+
| 101  | 3-105 |     90 |
| 102  | 3-105 |     91 |
| 103  | 3-105 |     92 |
| 104  | 3-105 |     89 |
| 105  | 3-105 |     88 |
+------+-------+--------+
```

- **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。

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

```
+-------+-------------+
| c_no  | AVG(degree) |
+-------+-------------+
| 3-105 |     87.6667 |
| 3-245 |     76.3333 |
| 6-166 |     81.6667 |
+-------+-------------+

-- 查询 score 表
select * from score;
a表
+------+-------+--------+
| s_no | c_no  | degree |
+------+-------+--------+
| 101  | 3-105 |     90 |
| 102  | 3-105 |     91 |
| 103  | 3-105 |     92 |
| 103  | 3-245 |     86 |
| 103  | 6-166 |     85 |
| 104  | 3-105 |     89 |
| 105  | 3-105 |     88 |
| 105  | 3-245 |     75 |
| 105  | 6-166 |     79 |
| 109  | 3-105 |     76 |
| 109  | 3-245 |     68 |
| 109  | 6-166 |     81 |
+------+-------+--------+

-- 将表 b 作用于表 a 中查询数据
-- score a (b): 将表声明为 a (b),
-- 如此就能用 a.c_no = b.c_no 作为条件执行查询了。
SELECT * FROM score a WHERE degree < (
    (SELECT AVG(degree) FROM score b WHERE a.c_no = b.c_no)
);
+------+-------+--------+
| s_no | c_no  | degree |
+------+-------+--------+
| 105  | 3-245 |     75 |
| 105  | 6-166 |     79 |
| 109  | 3-105 |     76 |
```

```
| 109  | 3-245 |     68 |
| 109  | 6-166 |     81 |
+------+-------+--------+
```

- **子查询 - 4**

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

```
SELECT name, department FROM teacher WHERE no IN (SELECT t_no FROM course
+--------+-----------------+
| name   | department      |
+--------+-----------------+
| 李诚   | 计算机系        |
| 王萍   | 计算机系        |
| 刘冰   | 电子工程系      |
| 张旭   | 电子工程系      |
+--------+-----------------+
```

- **条件加组筛选**

查询 student 表中至少有 2 名男生的 class 。

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

```
select * from student where sex='男'
group by class
having count(class)>=2;
+-----+------+-----+-----------+-------+
| no  | name | sex | birthday  | class |
+-----+------+-----+-----------+-------+
| 101 | 曾华 | 男  | 1977-09-01 | 95033 |
| 102 | 匡明 | 男  | 1975-10-02 | 95031 |
+-----+------+-----+-----------+-------+
```

- **NOT LIKE 模糊查询取反**

查询 `student` 表中不姓 "王" 的同学记录。

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

- **YEAR 与 NOW 函数**

查询 `student` 表中每个学生的姓名和年龄。

```
select name, year(now())-year(birthday) as age
from student;
+-----------+------+
| name      | age  |
+-----------+------+
| 曾华      |   42 |
```

```
| 匡明       |   44 |
| 王丽       |   43 |
| 李军       |   43 |
| 王芳       |   44 |
| 陆军       |   45 |
| 王尼玛     |   43 |
| 张全蛋     |   44 |
| 赵铁柱     |   45 |
| 张飞       |   45 |
+------------+------+
```

- **MAX 与 MIN 函数**

查询 `student` 表中最大和最小的 `birthday` 值。

```
SELECT MAX(birthday), MIN(birthday) FROM student;
+---------------+---------------+
| MAX(birthday) | MIN(birthday) |
+---------------+---------------+
| 1977-09-01    | 1974-06-03    |
+---------------+---------------+
```

- **多段排序**

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

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

```
| 109 | 赵铁柱 | 男   | 1974-06-03 | 95031 |
+-----+--------+-----+------------+-------+
```

- **子查询 - 5**

查询 "男" 教师及其所上的课程。

```
SELECT * FROM course WHERE t_no in (SELECT no FROM teacher WHERE sex = '
+-------+--------------+------+
| no    | name         | t_no |
+-------+--------------+------+
| 3-245 | 操作系统     | 804  |
| 6-166 | 数字电路     | 856  |
+-------+--------------+------+
```

- **MAX 函数与子查询**

查询最高分同学的 score 表。

```
-- 找出最高成绩（该查询只能有一个结果）
SELECT MAX(degree) FROM score;

-- 根据上面的条件筛选出所有最高成绩表，
-- 该查询可能有多个结果，假设 degree 值多次符合条件。
SELECT * FROM score WHERE degree = (SELECT MAX(degree) FROM score);
+------+-------+--------+
| s_no | c_no  | degree |
+------+-------+--------+
| 103  | 3-105 |     92 |
+------+-------+--------+
```

- **子查询 - 6**

查询和 "李军" 同性别的所有同学 name 。

```
select name from student where sex = (
SELECT sex FROM student where name='李军'
);
```

```
+--------+
| name   |
+--------+
| 曾华   |
| 匡明   |
| 李军   |
| 陆军   |
| 王尼玛 |
| 张全蛋 |
| 赵铁柱 |
| 张飞   |
+--------+
```

- **子查询 - 7**

查询和 "李军" 同性别且同班的同学 `name` 。

```
SELECT name, sex, class FROM student WHERE sex = (
    SELECT sex FROM student WHERE name = '李军'
) AND class = (
    SELECT class FROM student WHERE name = '李军'
);
+----------+-----+-------+
| name     | sex | class |
+----------+-----+-------+
| 曾华     | 男  | 95033 |
| 李军     | 男  | 95033 |
| 王尼玛   | 男  | 95033 |
+----------+-----+-------+
```

- **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。

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

```
    SELECT no FROM student WHERE sex = '男'
);
+------+-------+--------+
| s_no | c_no  | degree |
+------+-------+--------+
| 101  | 3-105 |     90 |
| 102  | 3-105 |     91 |
| 104  | 3-105 |     89 |
| 109  | 3-105 |     76 |
+------+-------+--------+
```

- **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：

```
CREATE TABLE grade (
    low INT(3),
    upp INT(3),
    grade char(1)
);

INSERT INTO grade VALUES (90, 100, 'A');
INSERT INTO grade VALUES (80, 89, 'B');
INSERT INTO grade VALUES (70, 79, 'C');
INSERT INTO grade VALUES (60, 69, 'D');
INSERT INTO grade VALUES (0, 59, 'E');

SELECT * FROM grade;
+------+------+-------+
| low  | upp  | grade |
+------+------+-------+
|   90 |  100 | A     |
|   80 |   89 | B     |
|   70 |   79 | C     |
|   60 |   69 | D     |
|    0 |   59 | E     |
+------+------+-------+
```

- **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。

```
SELECT s_no, c_no, grade FROM score, grade
WHERE degree BETWEEN low AND upp;
+------+-------+-------+
| s_no | c_no  | grade |
+------+-------+-------+
| 101  | 3-105 | A     |
| 102  | 3-105 | A     |
| 103  | 3-105 | A     |
| 103  | 3-245 | B     |
| 103  | 6-166 | B     |
| 104  | 3-105 | B     |
| 105  | 3-105 | B     |
| 105  | 3-245 | C     |
| 105  | 6-166 | C     |
| 109  | 3-105 | C     |
| 109  | 3-245 | D     |
| 109  | 6-166 | B     |
+------+-------+-------+
```

- **连接查询**

准备用于测试连接查询的数据：

```
CREATE DATABASE testJoin;

CREATE TABLE person (
    id INT,
    name VARCHAR(20),
    cardId INT
);

CREATE TABLE card (
    id INT,
    name VARCHAR(20)
```

```
);

INSERT INTO card VALUES (1, '饭卡'), (2, '建行卡'), (3, '农行卡'), (4, '工商
SELECT * FROM card;
+------+-----------+
| id   | name      |
+------+-----------+
|    1 | 饭卡      |
|    2 | 建行卡    |
|    3 | 农行卡    |
|    4 | 工商卡    |
|    5 | 邮政卡    |
+------+-----------+

INSERT INTO person VALUES (1, '张三', 1), (2, '李四', 3), (3, '王五', 6);
SELECT * FROM person;
+------+--------+--------+
| id   | name   | cardId |
+------+--------+--------+
|    1 | 张三   |      1 |
|    2 | 李四   |      3 |
|    3 | 王五   |      6 |
+------+--------+--------+
```

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。如果设置了的话，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。

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

```
|    1 | 张三   |       1 |    1 | 饭卡       |
|    2 | 李四   |       3 |    3 | 农行卡     |
+------+--------+---------+------+------------+

-- 将 INNER 关键字省略掉，结果也是一样的。
-- 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 。

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

- **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 。

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

- **全外链接**

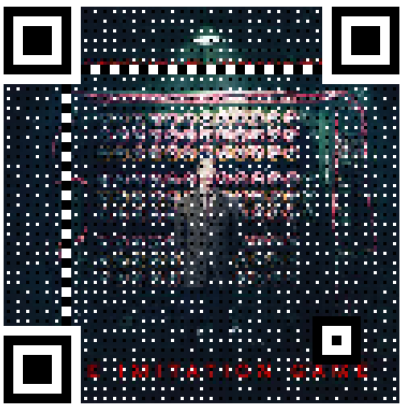
完整显示两张表的全部数据。

```
-- MySQL 不支持这种语法的全外连接
-- SELECT * FROM person FULL JOIN card on person.cardId = card.id;
-- 出现错误:
-- ERROR 1054 (42S22): Unknown column 'person.cardId' in 'on clause'

-- MySQL全连接语法，使用 UNION 将两张表合并在一起。
SELECT * FROM person LEFT JOIN card on person.cardId = card.id
UNION
SELECT * FROM person RIGHT JOIN card on person.cardId = card.id;
+------+--------+--------+------+-----------+
| id   | name   | cardId | id   | name      |
+------+--------+--------+------+-----------+
|    1 | 张三   |      1 |    1 | 饭卡      |
|    2 | 李四   |      3 |    3 | 农行卡    |
|    3 | 王五   |      6 | NULL | NULL      |
| NULL | NULL   |   NULL |    2 | 建行卡    |
| NULL | NULL   |   NULL |    4 | 工商卡    |
| NULL | NULL   |   NULL |    5 | 邮政卡    |
+------+--------+--------+------+-----------+
```

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！”