MySQL数据库(4)[1]

多表查询

多表关系

项目开发中，在进行数据库表结构设计时，会根据业务需求及业务模块之间的关系，分析并设计表结构，由于业务之间相互关联，所以各个表结构之间也存在着各种联系，基本上分为三种：

- 一对多(多对一)
- 多对多
- 一对一

一对多

案例: 部门 与 员工的关系

关系: 一个部门对应多个员工，一个员工对应一个部门

实现: 在多的一方建立外键，指向一方的主键

员工表(emp) N

id	name	age	dept_id
1	张无忌	20	1
2	杨逍	33	1
3	赵敏	18	2
4	常遇春	43	2

1 部门表(dept)

id	name
1	研发部
2	市场部
3	财务部
4	销售部

多对多

案例: 学生 与 课程的关系

关系: 一个学生可以选修多门课程，一门课程也可以供多个学生选择

实现: 建立第三张中间表，中间表至少包含两个外键，分别关联两方主键

id	name	no
1	黛绮丝	2000100101
2	谢逊	2000100102
3	殷天正	2000100103
4	韦一笑	2000100104

学生表(student) N

id	studentid	courseid
1	1	1
2	1	2
3	1	3
4	2	1
5	2	4

学生课程关系表(student_course)

id	name
1	Java
2	PHP
3	MySQL
4	Hadoop

N 课程表(course)

一对一

案例: 用户 与 用户详情的关系

关系: 一对一关系，多用于单表拆分，将一张表的基础字段放在一张表中，其他详情字段放在另一张表中，以提升操作效率

实现: 在任意一方加入外键，关联另外一方的主键，并且设置外键为唯一的(UNIQUE)

id	name	age	gender	phone
1	黄渤	45	1	18800001111
2	冰冰	35	2	18800002222
3	码云	55	1	18800008888
4	李彦宏	50	1	18800009999

用户基本信息表(tb_user)

id	degree	major	primaryschool	middleschool	university	userid
1	本科	舞蹈	静安区第一小学	静安区第一中学	北京舞蹈学院	1
2	硕士	表演	朝阳区第一小学	朝阳区第一中学	北京电影学院	2
3	本科	英语	杭州市第一小学	杭州市第一中学	杭州师范大学	3
4	本科	应用数学	阳泉第一小学	阳泉区第一中学	清华大学	4

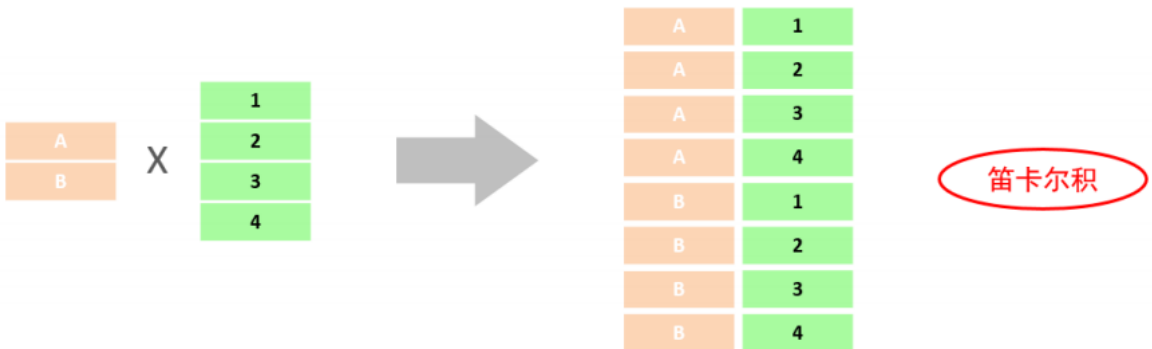
用户教育信息表(tb_user_edu)

> 多表查询就是指从多张表中查询数据 `SELECT * FROM goods,category;`

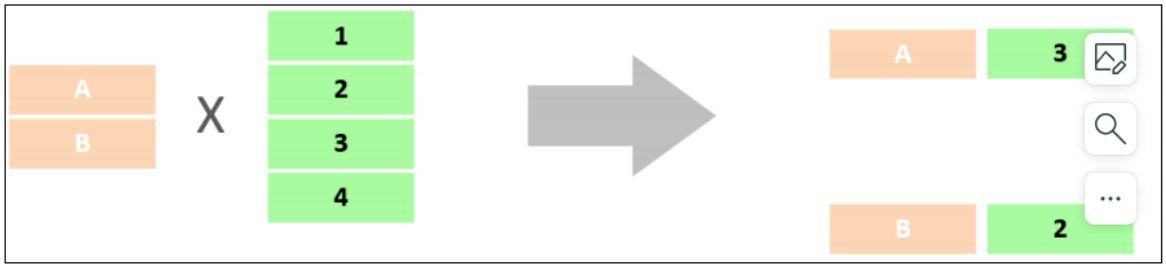
笛卡尔乘积现象

表查询中的笛卡尔乘积现象：多行表在查询时，如果定义了无效连接或者漏写了连接条件，就会产生笛卡尔乘积现象，所谓的笛卡尔乘积及时每个表的每一行都和其他表的每一行组合。**笛卡尔乘积现象**

笛卡尔积: 笛卡尔乘积是指在数学中，两个集合A集合 和 B集合的所有组合情况。



而在多表查询中，我们是需要消除无效的笛卡尔积的，只保留两张表关联部分的数据。



在SQL语句中，如何来去除无效的笛卡尔积呢？ 我们可以给多表查询加上连接查询的条件即可。

等值连接查询

通常是在存在主键外键关联关系的表之间的连接进行，使用"="连接相关的表

> n个表进行等值连接查询，最少需要n-1个等值条件来约束

```
-- 查询每个分类的所有商品信息
SELECT category.name,goods.name FROM goods,category WHERE
goods.category_no = category.no;
```

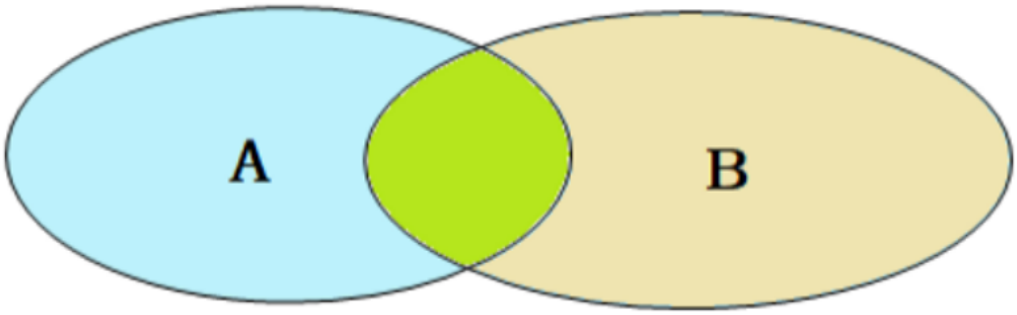
> 表的别名:
>
> ①. tableA as 别名1 , tableB as 别名2 ;
>
> ②. tableA 别名1 , tableB 别名2 ;
>
> 注意：一旦为表起了别名，就不能再使用表名来指定对应的字段了，此时只能够使用别名来指定字段。

自连接查询

多表查询不仅可以在多个表之间进行查询，也可以在一个表之中进行多表查询

内连接查询

内连接查询使用 `inner join` 关键字实现， `inner` 可以省略。内连接查询时，条件用 `on` 连接，多个条件使用 `()` 将其括起来.



内连接查询的是两张表交集部分的数据。(也就是绿色部分的数据)

```
--查询每个种类的所有商品
select c.name, g.name from goods g join category c on c.no =
g.category_no;
```

和等值查询差不多

外连接

外连接分为左外连接（ `left outer join` ） 和右外连接（ `right outer join` ）其值 `outer` 可以省略。外连接查询时，条件用 `on` 连接，多个条件使用 `()` 将其括起来.左外连接表示以左表为主表，右外连接表示以右表为主表。查询时将主表信息在从表中进行匹配

H5 左外连接

左外连接相当于查询表1(左表)的所有数据，当然也包含表1和表2交集部分的数据。

```
SELECT 字段列表 FROM 表1 LEFT [ OUTER ] JOIN 表2 ON 条件 ... ;
select 字段名 from 表1 left [outer] join 表2 on 条件 ... ;
```

H5 右外连接

右外连接相当于查询表2(右表)的所有数据，当然也包含表1和表2交集部分的数据。

```
SELECT 字段列表 FROM 表1 RIGHT [ OUTER ] JOIN 表2 ON 条件 ... ;
select 字段列表 from 表1 right [outer] join 表2 on 条件 ... ;
```

注意事项：

> 左外连接和右外连接是可以相互替换的，只需要调整在连接查询时SQL中，表结构的先后顺序就可以了。而我们在日常开发使用时，更偏向于左外连接。

子查询 [应用]

SQL语句中嵌套SELECT语句，称为嵌套查询，又称子查询。相对于子查询来说，在外部直接执行的查询语句被称作主查询。

子查询分为：

- 单列子查询: 返回单行单列数据的子查询
- 单行子查询: 返回单行多列数据的子查询
- 多行子查询: 返回数据是多行单列的数据
- 关联子查询: 子查询中如果使用了外部主SQL中的表或列，就说这个子查询跟外部SQL是相关的

单行子查询

单行单列

```
-- 查询零食种类的所有商品信息
select * from goods where category_no = (select no from category where name = '零食');

-- 查询部门名称是 ‘ ACCOUNTING’ 的所有员工信息
select * from emp where deptno = (select deptno from dept where dname = 'ACCOUNTING');
```

多列子查询

单行多列

```
-- 查询公司中和员工***相同薪水和奖金的员工
select * from emp e1 where (e1.sal,e1.comm) = (select e2.sal,e2.comm from emp e2 where e2.ename = '张青');
```

多行子查询

如果子查询返回了多行记录，则称这样的嵌套查询为多行子查询，多行子查询就需要用到多行记录的操作符

如： `in` , `all` , `any` , `not in`

IN : 在指定的集合范围之内，多选一

NOT IN : 不在指定的集合范围之内

ANY : 子查询返回列表中，有任意一个满足即可

ALL : 子查询返回列表的所有值都必须满足

```
-- 统计所有的员工分布在那些部门的信息
select * from category where category.no in (select category_no
from goods)
-- 查询公司中比任意一个员工的工资高的所有员工
select * from emp e1 where e1.sal > any (select e1.sal from emp
e2);
-- 查询公司中比所有的 Stock Clerk 工资高但不在 33部门 的员工
select * from emp e1 where e1.sal > all(select e2.sal from e2.emp
where w2.joblike '%助理');
```

- `>any` 表示大于子查询中的任意一个值，即大于最小值
- `>all` 表示大于子查询中的所有值，即大于最大的值

关联子查询

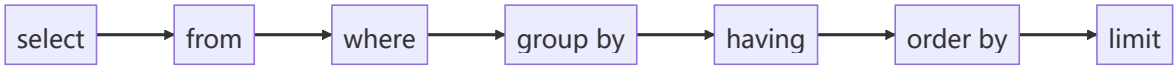
关联子查询与外部查询（主查询）之间存在联系，并且内部子查询的结果依赖于外部查询的值。换句话说，内部子查询的执行取决于外部查询的每一行。

```
SELECT student_name,
    (SELECT score FROM scores WHERE subject = 'Math' AND
students.student_id scores.student_id) AS math_score
FROM students;
```

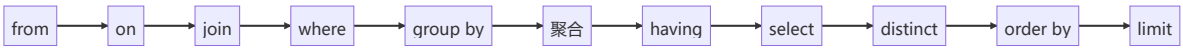
综合查询

```
-- 统计每个部门入职日期在 1980-1-1 日之后的人数，按照人数降序展示，展示第一
页，每页显示5条数据
SELECT DISTINCT emp.deptno FROM emp JOIN dept ON emp.deptno =
dept.deptno
WHERE HIREDATE ≥ '1980-01-01' GROUP BY emp.deptno
HAVING count(*) ≥ 2 ORDER BY count(*) DESC LIMIT 0, 5;
```

`SQL` 语句的书写顺序：



`SQL`语句的执行顺序：



单行函数

字符串函数

函数	实例	结果	描述
upper	select upper('sdfd');	SDFD	将字 母转 换为 大写
lower	select lower('ABc');	abc	将字 母转 换为 小写
concat	select concat('hello ','world');	'hello world'	字符 串连 接
substr/substring	select substr('hello world',2，4);	'ello'	截取 字符 串
length	select length('hello world');	11	获取 字符 串长 度
instr	select instr('hello world','world');	7	获取 子字 符串 在父 字符 串中 的索引
trim	select trim(' hello ');	'hello'	去除 两端 空格
ltrim	select ltrim(' hello');	'hello'	去掉 左端 的空 格
rtrim	select rtrim('hello ');	'hello'	去掉 右端 的空 格
replace	select replace('hello java','java','world');	'hello world'	替换 文本

数学函数

函数	示例	结果	作用
round(x,[y])	select round(5.64,1);	5.6	对指定的值进行四舍五入 是可以指定数值位数y
truncate	select truncate(5.6,0);	5	对指定的数及进行截取操 作，指定保留位数y
ceil(x)	select ceil(4.56)	5	返回不小于指定的值x得最 小整数，向上取整
floor(x)	select floor (8.5);	8	返回不大于指定的值x的最 大整数，向下取整
abs(x)	select abs(-12);	12	取绝对值

日期函数

函数	示例	结果	作用
current_timestamp()	select current_timestamp();	2019-11-07 20:53:47	获取当前时间戳
current_date()/CURDATE()	select current_date();	2019-11-07	获取当前日期
current_time()/CURTIME()	select current_time();	20:56:00	获取当前时间
now()	select now();	2019-11-07 20:57:15	获取当前时间+日期
ADDDATE(d,n)	SELECT ADDDATE("2017-06-15", INTERVAL 10 DAY);	2017-06-25	计算起始日期 d 加上 n 天的日期
ADDTIME(t,n)	SELECT ADDTIME('2011-11-11 11:11:11', 5);	2011-11-11 11:11:16	n 是一个时间表达式，时间 t 加 上时间表达式 n
DATEDIFF(d1,d2)	SELECT DATEDIFF('2001-01-01','2001-02-02')	-32	计算日期 d1-d2 之间相隔的天数
DATE_ADD(d，INTERVAL expr type)	SELECT DATE_ADD("2017-06-15", INTERVAL 10 DAY);	2017-06-25	计算起始日期 d 加上一个时间段 后的日期。type 值
DATE_SUB(date,INTERVAL expr type)	SELECT DATE_SUB("2017-06-15", INTERVAL 10 DAY);	2017-06-05	计算起始日期 d 减上一个时间段 后的日期。type 值
DATE_FORMAT(d,f)	SELECT DATE_FORMAT('2011-11-11 11:11:11','%Y-%m-%d %r')	2011-11-11 11:11:11 AM	按表达式 f的要求显示日期 d
DAY(d)	SELECT DAY("2017-06- 15")	15	返回日期值 d 的日期部分

函数	示例	结果	作用
DAYNAME(d)	SELECT DAYNAME('2011-11-11 11:11:11')	Friday	返回日期 d 是星期几，如 Monday,Tuesday
DAYOFMONTH(d)	SELECT DAYOFMONTH('2011-11- 11 11:11:11')	11	计算日期 d 是本月的第几天
DAYOFWEEK(d)	SELECT DAYOFWEEK('2011-11- 11 11:11:11')	6	日期 d 今天是星期几，1 星期日，2 星期一，以此类推
DAYOFYEAR(d)	SELECT DAYOFYEAR('2011-11- 11 11:11:11')	315	计算日期 d 是本年的第几天
EXTRACT(type FROM d)	SELECT EXTRACT(MINUTE FROM '2011-11-11 11:11:11')	11	从日期 d 中获取指定的值，type 指定返回的 值。
FROM_DAYS(n)	SELECT FROM_DAYS(1111)	0003-01-16	计算从 0000 年 1 月 1 日开始 n 天后的日期
HOUR(t)	SELECT HOUR('1:2:3')	1	返回 t 中的小时 值
LAST_DAY(d)	SELECT LAST_DAY("2017-06-20");	2017-06-30	返回给给定日期 的那一月份的最 后一天
MAKEDATE(year, day-ofyear)	SELECT MAKEDATE(2017, 3);	2017-01-03	基于给定参数年 份 year 和所在 年中的天数序号 day-of-year 返回 一个日期
MINUTE(t)	SELECT MINUTE('1:2:3')	2	返回 t 中的分钟 值
MONTHNAME(d)	SELECT MONTHNAME('2011-11- 11 11:11:11')	November	返回日期当中的 月份名称，如 November

函数	示例	结果	作用
MONTH(d)	SELECT MONTH('2011- 11-11 11:11:11')	11	返回日期d中的月份值，1 到 12
QUARTER(d)	SELECT QUARTER('2011-11-11 11:11:11')	4	返回日期d是第几季节，返回 1 到 4
SECOND(t)	SELECT SECOND('1:2:3')	3	返回 t 中的秒钟 值
SUBDATE(d,n)	SELECT SUBDATE('2011-11-11 11:11:11', 1)	2011-11-10 11:11:11	日期 d 减去 n 天后的日期
WEEK(d)	SELECT WEEK('2011-11-11 11:11:11')	45	计算日期 d 是本年的第几个星 期，范围是 0 到 53
WEEKDAY(d)	SELECT WEEKDAY("2017-06-15");	3	日期 d 是星期几，0 表示星期一，1 表示星期二
WEEKOFYEAR(d)	SELECT WEEKOFYEAR('2011-11- 11 11:11:11')	45	计算日期 d 是本年的第几个星 期，范围是 0 到 53
YEARWEEK(date, mode)	SELECT YEARWEEK("2017-06- 15");	201724	返回年份及第几 周（0到53）， mode 中 0 表示 周天，1 表示周 一，以此类推

type 值可以是:

- MICROSECOND 微秒
- SECOND 秒
- MINUTE 分钟
- HOUR 小时
- DAY 天

- WEEK 周
- MONTH 月
- QUARTER 季度
- YEAR 年
- MINUTE_SECOND 分钟:秒
- HOUR_SECOND 小时:分钟 : 秒
- HOUR_MINUTE 小时:分钟
- DAY_SECOND 天 小时:分钟:秒
- DAY_MINUTE 天 小时:分钟
- DAY_HOUR 天 小时
- YEAR_MONTH 年-月

可以被用在format字符串:

- %M 月名字(January……December)
- %W 星期名字(Sunday……Saturday)
- %D 有英语前缀的月份的日期(1st, 2nd, 3rd, 等等。)
- %Y 年, 数字, 4 位
- %y 年, 数字, 2 位
- %a 缩写的星期名字(Sun……Sat)
- %d 月份中的天数, 数字(00……31)
- %e 月份中的天数, 数字(0……31)
- %m 月, 数字(01……12)
- %c 月, 数字(1……12)
- %b 缩写的月份名字(Jan……Dec)
- %j 一年中的天数(001……366)
- %H 小时(00……23)
- %k 小时(0……23)
- %h 小时(01……12)
- %I 小时(01……12)
- %l 小时(1……12)
- %i 分钟, 数字(00……59)

- %r 时间,12 小时(hh:mm:ss [AP]M)
- %T 时间,24 小时(hh:mm:ss)
- %S 秒(00……59)
- %s 秒(00……59)
- %p AM或PM
- %w 一个星期中的天数(0=Sunday ……6=Saturday)
- %U 星期(0……52), 这里星期天是星期的第一天
- %u 星期(0……52), 这里星期一是星期的第一天

```
# 设置指定日期的时间间隔,date：表示输入的时间、value：表示间隔时间（正数表示在时间之前负数表示之后）
date_sub(date, INTERVAL value day/week/year)
```

补充

IFNULL 用法

MySQL **IFNULL** 函数是MySQL控制流函数之一，它接受两个参数，如果不是NULL， 则返回第一个参数。 否则，**IFNULL** 函数返回第二个参数。

```
-- 如果行数据num为null，查询结果中显示‘暂无’，如果不是null，显示num的值
SELECT IFNULL(num, '暂无') FROm 表名;
select ifnull(num, '暂无') from 表名;
```

> 应避免在 `WHERE` 子句中使用 `IFNULL` 函数，因为它会降低查询的性能。如 果要检查值是否为 `NULL` ，则可以在`WHERE`子句中使用 `IS NULL` 或 `IS NOT NULL` 。

CASE语句

`CASE` 语句遍历条件并在满足第一个条件时返回一个值（如`IF-THEN-ELSE` 语句）。 因此，一旦条件为真，它将停止读取并返回结果。 如果没有条件为真，它将返回 ELSE 子句中的值。 如果没有ELSE部分且没有条件为真，则返回NULL。

```
case
    when 条件1 then 条件为真返回的结果1
    when 条件2 then 结果2
    else 结果
end
```

```sql
# END 表示结束

SELECT CustomerName, City, Country
FROM Customers
ORDER BY
(CASE
    WHEN City IS NULL THEN Country
    ELSE City
END);

# 如果city是NULL 按照country排序
-- 按照成绩输出等级
SELECT id, score, CASE
    WHEN score < 60 THEN 'D'
    WHEN score < 70 THEN 'C'
    WHEN score < 80 THEN 'B'
    END '成绩等级' FROM score;
```

1. 第四天MySQL数据库学习笔记 ↩