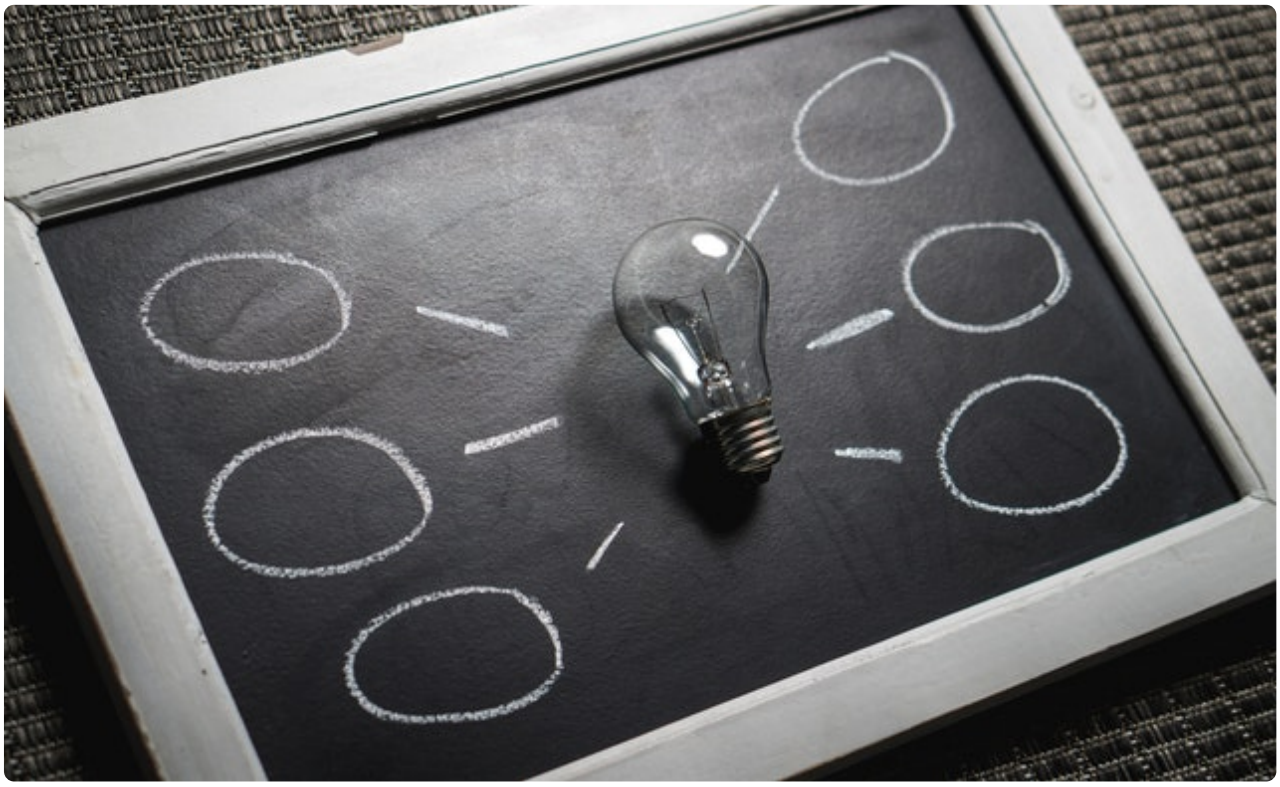
11 面试常见的高级查询 - 连接、联合、子查询

更新时间：2020-03-25 10:06:34



> 完成工作的方法，是爱惜每一分钟。——达尔文

MySQL 的面试里，索引、事务和复杂查询几乎是离不开的话题，而这里的复杂查询指的就是连接、联合和子查询。复杂查询并不常用（相对于单表形式的简单查询而言），所以，很多 ORM 框架对它的实现也并不优雅，也就自然而然的成为了工作、面试中的难题。这一节里，我将对复杂查询进行详细解读，相较于之前的内容，可能会让你觉得 “与众不同”，因为这一节几乎都是实战性质的应用。

在真正的去讲解这些高级（复杂）查询之前，我们需要先去做一些准备工作，想必你应该也猜到了：建表和 fake 一些数据（fake 与 mock 是相同的意思，只是我的个人习惯，忽略即可）。执行如下 SQL 语句：

```sql
-- 创建 imooc_user 表
CREATE TABLE `imooc_user` (
  `user_id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '用户 id',
  `name` char(64) NOT NULL DEFAULT '' COMMENT '姓名',
  `age` int(11) NOT NULL DEFAULT '0' COMMENT '年龄',
  PRIMARY KEY (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='慕课网用户信息表';

-- 创建 imooc_course 表
CREATE TABLE `imooc_course` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT 'id',
  `user_id` bigint(20) NOT NULL COMMENT '用户 id',
  `cname` char(64) NOT NULL DEFAULT '' COMMENT '课程名',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='慕课网课程信息表';

-- 插入数据到 imooc_user 表
INSERT INTO `imooc_user` (`user_id`, `name`, `age`) VALUES (1, 'qinyi', 19);
INSERT INTO `imooc_user` (`user_id`, `name`, `age`) VALUES (2, 'abc', 32);
INSERT INTO `imooc_user` (`user_id`, `name`, `age`) VALUES (3, 'xyz', 30);
INSERT INTO `imooc_user` (`user_id`, `name`, `age`) VALUES (4, 'mno', 29);

-- 插入数据到 imooc_course 表
INSERT INTO `imooc_course` (`id`, `user_id`, `cname`) VALUES (1, 1, '广告系统');
INSERT INTO `imooc_course` (`id`, `user_id`, `cname`) VALUES (2, 1, '优惠券系统');
INSERT INTO `imooc_course` (`id`, `user_id`, `cname`) VALUES (3, 1, '卡包系统');
INSERT INTO `imooc_course` (`id`, `user_id`, `cname`) VALUES (4, 2, 'Java');
INSERT INTO `imooc_course` (`id`, `user_id`, `cname`) VALUES (5, 2, 'Python');
INSERT INTO `imooc_course` (`id`, `user_id`, `cname`) VALUES (6, 3, 'MySQL');
INSERT INTO `imooc_course` (`id`, `user_id`, `cname`) VALUES (7, 5, 'Linux');
```

我们创建了两张数据表：imooc_user 和 imooc_course，你可以很容易发现 imooc_course 表中的 user_id 是 imooc_user 表的逻辑外键。且插入的数据里面，两张表各自有对方不存在的记录（仔细看看，你会发现的），这当然也是为后文做准备。

1 出现频率最高的连接查询

我们最常遇到的复杂查询就是连接查询了，它是将多个表（绝大多数情况下是两张表）联合起来查询，连接的方式一共有四种：内连接、外连接、自然连接和交叉连接。连接查询的最终目的是在一次查询中获取多张表的数据。下面，开始我们的连接查询之旅吧。

1.1 内连接

内连接是有条件匹配的连接，多个表之间依据给定的条件进行连接，并保留符合匹配结果的记录。以两张表的连接举例，它所表达的含义是：从左表（SQL 语句中位于左边的表）中取出一条记录，与右表中的所有记录去匹配，保留匹配成功的记录，并拼接打印。它的语法如下所示：

```sql
SELECT col1, col2 FROM left [INNER] JOIN right ON left.colx = right.coly
```

举个例子来看看吧，我想要查询 imooc_user 表对应的 imooc_course 信息，可以这样：

```
-- 使用 INNER JOIN 连接右表
mysql> SELECT * FROM imooc_user AS user INNER JOIN imooc_course AS course ON user.user_id = course.user_id;
+---------+-------+-----+----+---------+-----------------+
| user_id | name  | age | id | user_id | cname           |
+---------+-------+-----+----+---------+-----------------+
|       1 | qinyi |  19 |  1 |       1 | 广告系统        |
|       1 | qinyi |  19 |  2 |       1 | 优惠券系统      |
|       1 | qinyi |  19 |  3 |       1 | 卡包系统        |
|       2 | abc   |  32 |  4 |       2 | Java            |
|       2 | abc   |  32 |  5 |       2 | Python          |
|       3 | xyz   |  30 |  6 |       3 | MySQL           |
+---------+-------+-----+----+---------+-----------------+

-- 使用 JOIN 连接右表，与 INNER JOIN 结果一致
mysql> SELECT * FROM imooc_user AS user JOIN imooc_course AS course ON user.user_id = course.user_id;
+---------+-------+-----+----+---------+-----------------+
| user_id | name  | age | id | user_id | cname           |
+---------+-------+-----+----+---------+-----------------+
|       1 | qinyi |  19 |  1 |       1 | 广告系统        |
|       1 | qinyi |  19 |  2 |       1 | 优惠券系统      |
|       1 | qinyi |  19 |  3 |       1 | 卡包系统        |
|       2 | abc   |  32 |  4 |       2 | Java            |
|       2 | abc   |  32 |  5 |       2 | Python          |
|       3 | xyz   |  30 |  6 |       3 | MySQL           |
+---------+-------+-----+----+---------+-----------------+
```

可以看到，连接查询默认（SELECT *）会返回两张表的所有列，如果我们不需要，可以单独指定想要的列。ON 子句的功能与 WHERE 子句的功能是类似的，用于条件匹配，当然，ON 子句之后也可以指定单表的条件。例如：

```
SELECT * FROM imooc_user AS user JOIN imooc_course AS course ON user.user_id = course.user_id AND user.user_id <= 1
```

1.2 外连接

从内连接的查询结果可以看出，它只会保留两个表中完全匹配的记录，而外连接则不同，不论 “主表” 符不符合匹配条件，记录都将被保留。而 “主表” 的依据则是外连接的两种方式：左外连接和右外连接。左外连接保留左表，即此时左表是 “主表”，右外连接则刚好相反。外连接的语法如下：

```
-- 左外连接
SELECT col1, col2 FROM left LEFT JOIN right ON left.colx = right.coly

-- 右外连接
SELECT col1, col2 FROM left RIGHT JOIN right ON left.colx = right.coly
```

语法层面上看，与内连接几乎是一样的，只是把 INNER 换成了 LEFT/RIGHT。同样，我们还是看两个例子：

```
-- 左外连接
mysql> SELECT * FROM imooc_user AS user LEFT JOIN imooc_course AS course ON user.user_id = course.user_id;
+---------+-------+-----+------+---------+-----------------+
| user_id | name  | age | id   | user_id | cname           |
+---------+-------+-----+------+---------+-----------------+
|       1 | qinyi |  19 |    1 |       1 | 广告系统        |
|       1 | qinyi |  19 |    2 |       1 | 优惠券系统      |
|       1 | qinyi |  19 |    3 |       1 | 卡包系统        |
|       2 | abc   |  32 |    4 |       2 | Java            |
|       2 | abc   |  32 |    5 |       2 | Python          |
|       3 | xyz   |  30 |    6 |       3 | MySQL           |
|       4 | mno   |  29 | NULL |    NULL | NULL            |
+---------+-------+-----+------+---------+-----------------+

-- 右外连接
mysql> SELECT * FROM imooc_user AS user RIGHT JOIN imooc_course AS course ON user.user_id = course.user_id;
+---------+-------+------+----+---------+-----------------+
| user_id | name  | age  | id | user_id | cname           |
+---------+-------+------+----+---------+-----------------+
|       1 | qinyi |  19  |  1 |       1 | 广告系统        |
|       1 | qinyi |  19  |  2 |       1 | 优惠券系统      |
|       1 | qinyi |  19  |  3 |       1 | 卡包系统        |
|       2 | abc   |  32  |  4 |       2 | Java            |
|       2 | abc   |  32  |  5 |       2 | Python          |
|       3 | xyz   |  30  |  6 |       3 | MySQL           |
|    NULL | NULL  | NULL |  7 |       5 | Linux           |
+---------+-------+------+----+---------+-----------------+
```

正如之前所说，即使没有匹配，主表也会保留记录。但是，由于此时副表（相对于主表）不能完成匹配，就会以 NULL 返回。内连接和外连接不仅非常相似，它们也几乎包揽了连接查询的话题，所以，理解和应用就从现在开始吧。

1.3 自然连接

自然连接的含义是自动匹配连接条件，MySQL 自动的以字段名作为匹配模式，同名的字段（这也就是为什么我在创建 imooc_user 表时，指定主键的名称是 user_id，目的就是为了与 imooc_course 相对应）就会作为条件，不论是几个。自然连接分为自然内连接和自然外连接，语法及说明如下：

```
-- 自然内连接，与内连接类似，但是不提供连接条件
SELECT col1, col2 FROM left NATURAL JOIN right

-- 自然外连接，与外连接类似，但是不提供连接条件
SELECT col1, col2 FROM left NATURAL LEFT/RIGHT JOIN right
```

自然连接的思想和使用方法都是非常简单的，但是，由于它要求列名需要一致（相同的列名也许不能作为匹配条件），可用性并不高。下面，给出几个简单的示例（大多数情况下，知道在 MySQL 中存在自然连接这个概念就可以）：

```
-- 自然内连接
mysql> SELECT * FROM imooc_user NATURAL JOIN imooc_course;
+---------+-------+-----+----+-----------------+
| user_id | name  | age | id | cname           |
+---------+-------+-----+----+-----------------+
|       1 | qinyi |  19 |  1 | 广告系统        |
|       1 | qinyi |  19 |  2 | 优惠券系统      |
|       1 | qinyi |  19 |  3 | 卡包系统        |
|       2 | abc   |  32 |  4 | Java            |
|       2 | abc   |  32 |  5 | Python          |
|       3 | xyz   |  30 |  6 | MySQL           |
+---------+-------+-----+----+-----------------+

-- 自然左外连接
mysql> SELECT * FROM imooc_user NATURAL LEFT JOIN imooc_course;
+---------+-------+-----+------+-----------------+
| user_id | name  | age | id   | cname           |
+---------+-------+-----+------+-----------------+
|       1 | qinyi |  19 |    1 | 广告系统        |
|       1 | qinyi |  19 |    2 | 优惠券系统      |
|       1 | qinyi |  19 |    3 | 卡包系统        |
|       2 | abc   |  32 |    4 | Java            |
|       2 | abc   |  32 |    5 | Python          |
|       3 | xyz   |  30 |    6 | MySQL           |
|       4 | mno   |  29 | NULL | NULL            |
+---------+-------+-----+------+-----------------+

-- 自然右外连接
mysql> SELECT * FROM imooc_user NATURAL RIGHT JOIN imooc_course;
+---------+----+-----------------+-------+------+
| user_id | id | cname           | name  | age  |
+---------+----+-----------------+-------+------+
|       1 |  1 | 广告系统        | qinyi |   19 |
|       1 |  2 | 优惠券系统      | qinyi |   19 |
|       1 |  3 | 卡包系统        | qinyi |   19 |
|       2 |  4 | Java            | abc   |   32 |
|       2 |  5 | Python          | abc   |   32 |
|       3 |  6 | MySQL           | xyz   |   30 |
|       5 |  7 | Linux           | NULL  | NULL |
+---------+----+-----------------+-------+------+
```

1.4 交叉连接

相对于之前的有条件连接，交叉连接也被称为“无条件连接”，它会将左表的每一条记录与右表的每一条记录进行连接，结果中的列数等于两表列数之和，行数等于两表行数之积。也就是说交叉连接的结果是“笛卡尔积”。它的语法如下所示：

```
SELECT col1, col2 FROM left CROSS JOIN right
```

需要注意，由于交叉连接的结果是两张表记录的乘积，所以，返回的数据会特别多，不应该轻易执行这样的操作。实际上，在我们的大多数应用中，笛卡尔积并没有意义，所以，出现的概率极低，有兴趣的话可以执行如下语句查看结果（返回数据太多，且没有实际意义，我这里不贴出输出）。

```
SELECT * FROM imooc_user CROSS JOIN imooc_course
```

连接查询是 MySQL 高级查询中的重中之重，以至于我经常听到朋友说：用了 MySQL 这么些年，高级查询也就只用过连接。其实事实也真的就是这样，连接查询不仅面向众多的业务需求，而且难度也适中，所以，市场自然也就会更广阔一些。

2 常常被忽略的联合查询

联合查询的关键字是 UNION，它将两个或多个查询的结果拼接在一起返回。正是由于它会将不同的查询结果拼接在一起，所以，每一个查询结果的字段数必须是相同的。特殊的是，拼接过程并不会在意数据类型（但还是强烈推荐对应列的数据类型一致，否则，很容易在代码中出现类型错误）。下面，我们还是先来看一个简单的例子（并没有实际意义，只是作为示例说明）：

```
mysql> SELECT name, age FROM imooc_user UNION SELECT user_id, cname FROM imooc_course;
+-------+-----------------+
| name  | age             |
+-------+-----------------+
| qinyi | 19              |
| abc   | 32              |
| xyz   | 30              |
| mno   | 29              |
| 1     | 广告系统        |
| 1     | 优惠券系统      |
| 1     | 卡包系统        |
| 2     | Java            |
| 2     | Python          |
| 3     | MySQL           |
| 5     | Linux           |
+-------+-----------------+
```

可以看到，正如之前所述，两个表的查询结果拼接在一起返回。UNION 有个特性，它会去除重复的行（所有的字段都相同），如果想要完整的数据，则需要加上 ALL 选项，可以自行执行如下两条 SQL 语句并分析结果。

```
SELECT user_id FROM imooc_user UNION SELECT user_id FROM imooc_course
SELECT user_id FROM imooc_user UNION ALL SELECT user_id FROM imooc_course
```

联合查询有一个最容易出错的地方：对结果进行排序。且它会把多个查询结果进行拼接，单表结果排序与总体排序又该怎么办 ？首先，对于单个查询结果排序的话，需要将它的 SELECT 语句用括号括起来，同时配合 LIMIT 使用（如果不配合 LIMIT，会被语法分析器优化时去除，导致排序失效）。示例如下：

```
mysql> (SELECT name, age FROM imooc_user ORDER BY age DESC LIMIT 2) UNION SELECT user_id, cname FROM imooc_course;
+------+-----------------+
| name | age             |
+------+-----------------+
| abc  | 32              |
| xyz  | 30              |
| 1    | 广告系统        |
| 1    | 优惠券系统      |
| 1    | 卡包系统        |
| 2    | Java            |
| 2    | Python          |
| 3    | MySQL           |
| 5    | Linux           |
+------+-----------------+

-- 可以试试如下的联合查询，ORDER BY 之后并没有 LIMIT
(SELECT name, age FROM imooc_user ORDER BY age DESC) UNION SELECT user_id, cname FROM imooc_course
```

如果是针对最终的查询结果进行排序，在最后一个 SELECT 语句之后使用 ORDER BY 即可，这里示例我不再给出，我相信你可以写出来。

3 不易理解的子查询

子查询是嵌套在查询语句中的查询，按照它出现的位置（使用的关键字）可以分为三类：FROM 子查询、WHERE 子查询和 EXISTS 子查询。子查询可以包含普通 SELECT 可以包含的任何子句，例如：DISTINCT、GROUP BY、ORDER BY、LIMIT 等等。下面，我们就依次来看看这三类子查询的玩法。

3.1 FROM 子查询

这一类子查询是跟在 FROM 子句之后的，它的语义是：先查出“一张表”，再去查询这张表。等等，这是什么意思？为什么我没有看懂呢？莫慌，刚看到这句话，我也不明白。看个例子吧，虽然它本身没什么意义。

```
-- 报错的信息提示：子查询派生出的表必须要有个名字
mysql> SELECT name, age FROM (SELECT * FROM imooc_user WHERE name = 'qinyi');
ERROR 1248 (42000): Every derived table must have its own alias

-- 给子查询派生出的表指定 user 别名
mysql> SELECT name, age FROM (SELECT * FROM imooc_user WHERE name = 'qinyi') AS user;
+--------+-----+
| name   | age |
+--------+-----+
| qinyi  |  19 |
+--------+-----+
```

从这个例子里，我们就可以得出结论了：FROM 子查询实际上是得到了一张表，我们查询的就是这张派生表。FROM 子查询常常用于将复杂的查询分解，将复杂的查询条件拆解到多次查询中去。

3.2 WHERE 子查询

顾名思义，WHERE 子查询是跟在 WHERE 条件中的，它的语义是：先根据条件筛选，再根据条件查询。看到这句话的你，是不是接近崩溃了？不过，这事儿真不怪我，不信你就看看下面的例子。

```
-- WHERE 子查询结果是多个值，使用 IN 查询
mysql> SELECT * FROM imooc_user WHERE user_id IN (SELECT user_id FROM imooc_course WHERE user_id < 3);
+---------+--------+-----+
| user_id | name   | age |
+---------+--------+-----+
|       1 | qinyi  |  19 |
|       2 | abc    |  32 |
+---------+--------+-----+

-- WHERE 子查询结果是单个值，使用等于就可以（当然也可以用 IN）
mysql> SELECT * FROM imooc_user WHERE user_id = (SELECT user_id FROM imooc_course WHERE cname LIKE '广告%');
+---------+--------+-----+
| user_id | name   | age |
+---------+--------+-----+
|       1 | qinyi  |  19 |
+---------+--------+-----+
```

看到示例的你一定又明白了 WHERE 子查询是怎样的查询，其实，它也并不难理解。WHERE 子查询的常见用法是把其他表的查询结果当做当前表的查询条件进行二次查询，非常类似于 JOIN 的思想。

3.3 EXISTS 子查询

这一类子查询包裹在 EXISTS 关键字里面，它所表达的语义是：存在才触发。相对于 FROM 和 WHERE 子查询来说，EXISTS 子查询就容易理解多了。同样，我们还是看两个例子吧：

```
-- 存在课程名是 JAVA 的课程，所以，子查询存在
mysql> SELECT * FROM imooc_user WHERE EXISTS(SELECT * FROM imooc_course WHERE cname LIKE '%JAVA%') AND user_id = 1;
+---------+-------+-----+
| user_id | name  | age |
+---------+-------+-----+
|       1 | qinyi |  19 |
+---------+-------+-----+

-- 不存在课程名包含 PHP 的课程，所以，子查询不存在
mysql> SELECT * FROM imooc_user WHERE EXISTS(SELECT * FROM imooc_course WHERE cname LIKE '%PHP%') AND user_id = 1;
Empty set (0.00 sec)
```

单单看这三类子查询的定义肯定是不易理解的，甚至是不知所云的。所以，学习新的知识点，不要盲目的去看它的定义，一定要结合着实例去看。很多时候，高大上的名词背后仅仅是简单技术的包装。最后，由于子查询会让 SQL 语句变得太长，且使用场景相对有限，出场机会并不会太多。

4 总结

为了讲解清楚 MySQL 中的高级查询，这一节里我给出了很多示例，但是这些仍远远不够。之所以称它们是高级查询，因为真的很难，而且不容易记住。所以，多去设想一些场景（这也并不容易），并尝试使用这些高级查询解决你 fake 的难题，才能做到学以致用。另外，记住，学习的进程一定是温故而知新的，谁也不会有那么好的记性。

5 问题

关于连接查询，你在工作、面试中遇到过这些问题吗？你是怎么做的？

关于联合查询，你能想到一些合适的使用场景吗？

子查询可以改造为连接查询吗？怎么做呢？

6 参考资料

《高性能 MySQL（第三版）》

MySQL 官方文档：SELECT Statement

}

10 索引定义及其优化，你知道多少？

12 死锁是怎么出现的？又是怎么解决的呢？