

数据库开发-MySQL

1. 多表查询

1.1 概述

1.1.1 数据准备

SQL脚本:

```
1  #建议: 创建新的数据库
2  create database db04;
3  use db04;
4
5  -- 部门表
6  create table tb_dept
7  (
8      id          int unsigned primary key auto_increment comment '主键
      ID',
9      name        varchar(10) not null unique comment '部门名称',
10     create_time  datetime    not null comment '创建时间',
11     update_time  datetime    not null comment '修改时间'
12 ) comment '部门表';
13 -- 部门表测试
14 insert into tb_dept (id, name, create_time, update_time)
15 values (1, '学工部', now(), now()),
16        (2, '教研部', now(), now()),
17        (3, '咨询部', now(), now()),
18        (4, '就业部', now(), now()),
19        (5, '人事部', now(), now());
20
21 -- 员工表
22 create table tb_emp
23 (
24     id          int unsigned primary key auto_increment comment
      'ID',
25     username    varchar(20)      not null unique comment '用户名',
26     password    varchar(32) default '123456' comment '密码',
27     name        varchar(10)      not null comment '姓名',
```

```

28     gender        tinyint unsigned not null comment '性别, 说明: 1 男,
29     2 女',
29     image         varchar(300) comment '图像',
30     job           tinyint unsigned comment '职位, 说明: 1 班主任, 2 讲师,
31     3 学工主管, 4 教研主管, 5 咨询师',
31     entrydate     date comment '入职时间',
32     dept_id       int unsigned comment '部门ID',
33     create_time   datetime          not null comment '创建时间',
34     update_time   datetime          not null comment '修改时间'
35 ) comment '员工表';
36 -- 员工表测试数据
37 INSERT INTO tb_emp(id, username, password, name, gender, image, job,
38     entrydate, dept_id, create_time, update_time)
39 VALUES
40 (1, 'jinyong', '123456', '金庸', 1, '1.jpg', 4, '2000-01-01', 2, now(), now()),
41 (2, 'zhangwuji', '123456', '张无忌', 1, '2.jpg', 2, '2015-01-
42     01', 2, now(), now()),
43 (3, 'yangxiao', '123456', '杨逍', 1, '3.jpg', 2, '2008-05-
44     01', 2, now(), now()),
45 (4, 'weiyixiao', '123456', '韦一笑', 1, '4.jpg', 2, '2007-01-
46     01', 2, now(), now()),
47 (5, 'changyuchun', '123456', '常遇春', 1, '5.jpg', 2, '2012-12-
48     05', 2, now(), now()),
49 (6, 'xiaozhao', '123456', '小昭', 2, '6.jpg', 3, '2013-09-
50     05', 1, now(), now()),
51 (7, 'jixiaofu', '123456', '纪晓芙', 2, '7.jpg', 1, '2005-08-
52     01', 1, now(), now()),
53 (8, 'zhouzhiruo', '123456', '周芷若', 2, '8.jpg', 1, '2014-11-
54     09', 1, now(), now()),
55 (9, 'dingminjun', '123456', '丁敏君', 2, '9.jpg', 1, '2011-03-
56     11', 1, now(), now()),
57 (10, 'zhaomin', '123456', '赵敏', 2, '10.jpg', 1, '2013-09-
58     05', 1, now(), now()),
59 (11, 'luzhangke', '123456', '鹿杖客', 1, '11.jpg', 5, '2007-02-
60     01', 3, now(), now()),
61 (12, 'hebiweng', '123456', '鹤笔翁', 1, '12.jpg', 5, '2008-08-
62     18', 3, now(), now()),
63 (13, 'fangdongbai', '123456', '方东白', 1, '13.jpg', 5, '2012-11-
64     01', 3, now(), now()),
65 (14, 'zhangsanfeng', '123456', '张三丰', 1, '14.jpg', 2, '2002-08-
66     01', 2, now(), now()),
67 (15, 'yulianzhou', '123456', '俞莲舟', 1, '15.jpg', 2, '2011-05-
68     01', 2, now(), now()),

```

```

54 (16,'songyuanqiao','123456','宋远桥',1,'16.jpg',2,'2007-01-
    01',2,now(),now()),
55 (17,'chenyouliang','123456','陈友谅',1,'17.jpg',NULL,'2015-03-
    21',NULL,now(),now());

```

1.1.1.2 介绍

多表查询：查询时从多张表中获取所需数据

单表查询的SQL语句：`select 字段列表 from 表名;`

那么要执行多表查询，只需要使用逗号分隔多张表即可，如：`select 字段列表 from 表1, 表2;`

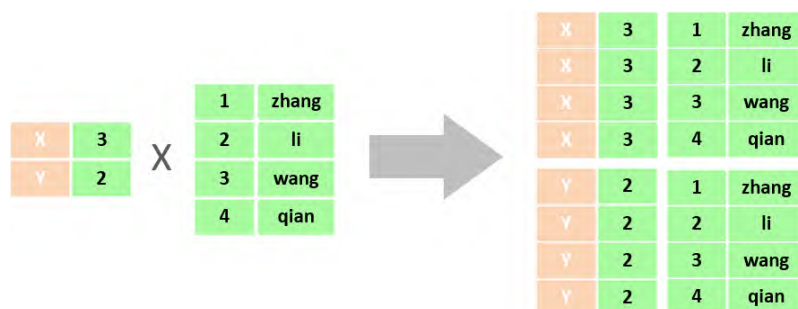
查询用户表和部门表中的数据：

```
1 select * from tb_emp , tb_dept;
```

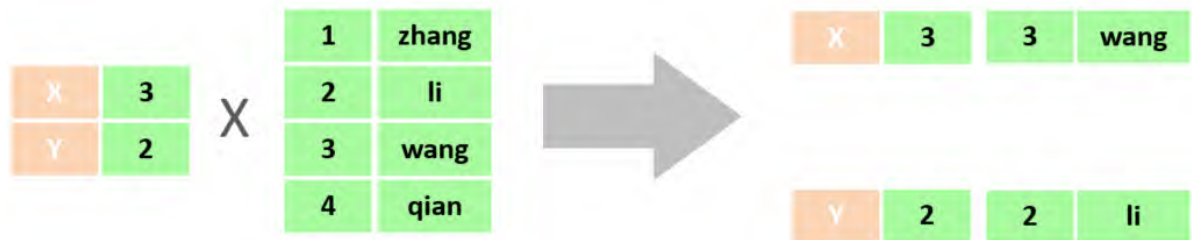
gender	image	job	entrydate	dept_id	emp.create_time	emp.update_time	dept.id	dept.name	dept.create_time
1	1.jpg	4	2006-01-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	5	人事部	2022-09-01 09:14:48
1	1.jpg	4	2006-01-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	4	技术部	2022-09-01 09:14:48
1	1.jpg	4	2006-01-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	3	市场部	2022-09-01 09:14:48
1	1.jpg	4	2006-01-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	2	财务部	2022-09-01 09:14:48
1	1.jpg	4	2006-01-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	1	生产部	2022-09-01 09:14:48
1	2.jpg	2	2015-01-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	5	人事部	2022-09-01 09:14:48
1	2.jpg	2	2015-01-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	4	技术部	2022-09-01 09:14:48
1	2.jpg	2	2015-01-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	3	市场部	2022-09-01 09:14:48
1	2.jpg	2	2015-01-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	2	财务部	2022-09-01 09:14:48
1	2.jpg	2	2015-01-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	1	生产部	2022-09-01 09:14:48
1	3.jpg	2	2008-05-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	5	人事部	2022-09-01 09:14:48
1	3.jpg	2	2008-05-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	4	技术部	2022-09-01 09:14:48
1	3.jpg	2	2008-05-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	3	市场部	2022-09-01 09:14:48
1	3.jpg	2	2008-05-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	2	财务部	2022-09-01 09:14:48
1	3.jpg	2	2008-05-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	1	生产部	2022-09-01 09:14:48
1	4.jpg	2	2007-01-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	5	人事部	2022-09-01 09:14:48
1	4.jpg	2	2007-01-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	4	技术部	2022-09-01 09:14:48
1	4.jpg	2	2007-01-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	3	市场部	2022-09-01 09:14:48
1	4.jpg	2	2007-01-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	2	财务部	2022-09-01 09:14:48
1	4.jpg	2	2007-01-01	2	2022-09-01 09:14:48	2022-09-01 09:14:48	1	生产部	2022-09-01 09:14:48
1	5.jpg	2	2012-12-05	2	2022-09-01 09:14:48	2022-09-01 09:14:48	5	人事部	2022-09-01 09:14:48
1	5.jpg	2	2012-12-05	2	2022-09-01 09:14:48	2022-09-01 09:14:48	4	技术部	2022-09-01 09:14:48
1	5.jpg	2	2012-12-05	2	2022-09-01 09:14:48	2022-09-01 09:14:48	3	市场部	2022-09-01 09:14:48

此时,我们看到查询结果中包含了大量的结果集, 总共85条记录, 而这其实就是员工表所有的记录(17行)与部门表所有记录(5行)的所有组合情况, 这种现象称之为笛卡尔积。

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。



在多表查询时，需要消除无效的笛卡尔积，只保留表关联部分的数据



在SQL语句中，如何去除无效的笛卡尔积呢？只需要给多表查询加上连接查询的条件即可。

```
1 select * from tb_emp , tb_dept where tb_emp.dept_id = tb_dept.id ;
```

	id	username	password	name	gender	image	job	entrydate	dept_id	create
6	6	xiaozhao	123456	小昭	2	6.jpg	3	2013-09-05	1	2022-01-01
7	7	jixiaofu	123456	纪晓芙	2	7.jpg	1	2005-08-01	1	2022-01-01
8	8	zhouzhiruo	123456	周芷若	2	8.jpg	1	2014-11-09	1	2022-01-01
9	9	dingminjun	123456	丁敏君	2	9.jpg	1	2011-03-11	1	2022-01-01
10	10	zhaomin	123456	赵敏	2	10.jpg	1	2013-09-05	1	2022-01-01
11	11	luzhangke	123456	鹿杖客	1	11.jpg	5	2007-02-01	3	2022-01-01
12	12	hebiweng	123456	鹤笔翁	1	12.jpg	5	2008-08-18	3	2022-01-01
13	13	fangdongbai	123456	方东白	1	13.jpg	5	2012-11-01	3	2022-01-01
14	14	zhangsanfeng	123456	张三丰	1	14.jpg	2	2002-08-01	2	2022-01-01
15	15	yulianzhou	123456	俞莲舟	1	15.jpg	2	2011-05-01	2	2022-01-01
16	16	songyuanqiao	123456	宋远桥	1	16.jpg	2	2007-01-01	2	2022-01-01

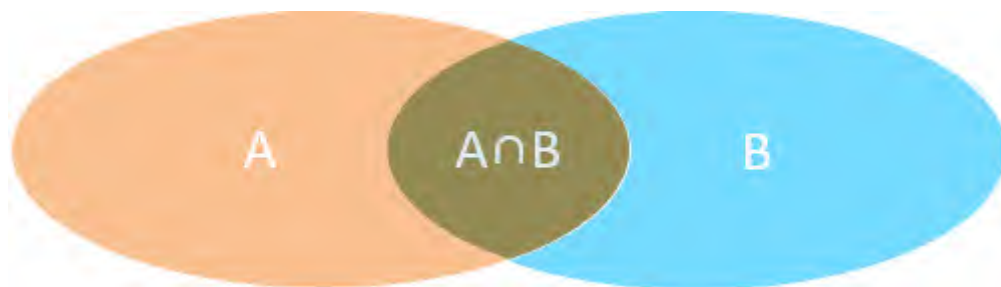
由于id为17的员工，没有dept_id字段值，所以在多表查询时，根据连接查询的条件并没有查询到。

1.1.1.3 分类

多表查询可以分为：

1. 连接查询

- 内连接：相当于查询A、B交集部分数据



2. 外连接

- 左外连接：查询左表所有数据 (包括两张表交集部分数据)
- 右外连接：查询右表所有数据 (包括两张表交集部分数据)

3. 子查询

1.2 内连接

内连接查询：查询两表或多表中交集部分数据。

内连接从语法上可以分为：

- 隐式内连接
- 显式内连接

隐式内连接语法：

```
1  select  字段列表  from  表1 , 表2  where  条件 ... ;
```

显式内连接语法：

```
1  select  字段列表  from  表1  [ inner ] join 表2  on  连接条件 ... ;
```

案例：查询员工的姓名及所属的部门名称

- 隐式内连接实现

```
1  select tb_emp.name , tb_dept.name -- 分别查询两张表中的数据
2  from tb_emp , tb_dept -- 关联两张表
3  where tb_emp.dept_id = tb_dept.id; -- 消除笛卡尔积
```

- 显式内连接实现

```
1  select tb_emp.name , tb_dept.name
2  from tb_emp inner join tb_dept
3  on tb_emp.dept_id = tb_dept.id;
```

	tb_emp.name	tb_dept.name
1	金庸	教研部
2	张无忌	教研部
3	杨逍	教研部
4	韦一笑	教研部
5	常遇春	教研部
6	小昭	学工部
7	纪晓芙	学工部
8	周芷若	学工部
9	丁敏君	学工部
10	赵敏	学工部
11	鹿杖客	咨询部
12	鹤笔翁	咨询部

多表查询时给表起别名：

- tableA as 别名1 , tableB as 别名2 ;
- tableA 别名1 , tableB 别名2 ;

书签

前言

目录

一、编程规约

二、异常日志

三、单元测试

四、安全规约

五、MySQL数据库

(一) 建表规约

(二) 索引规约

(三) SQL语句

(四) ORM映射

六、工程结构

七、设计规约

附1：版本历史

附2：专有名词解释

附3：错误码列表

9.【强制】对于数据库中表记录的查询和变更，只要涉及多个表，都需要在列名前加表的别名（或表名）进行限定。

34/51

Java 开发手册（黄山版）

说明：对多表进行查询记录、更新记录、删除记录时，如果对操作列没有限定表的别名（或表名），并且操作列在多个表中存在时，就会抛异常。

正例：select t1.name from first_table as t1 , second_table as t2 where t1.id = t2.id;

反例：在某业务中，由于多表关联查询语句没有加表的别名（或表名）的限制，正常运行两年后，最近在某个表中增加一个同名字段，在预发布环境做数据库变更后，线上查询语句出现出 1052 异常：Column 'name' in field list is ambiguous。

10.【推荐】SQL 语句中表的别名前加 as，并且以 t1、t2、t3、...的顺序依次命名。

说明：

1) 别名可以是表的简称，或者是依照表在 SQL 语句中出现的顺序，以 t1、t2、t3 的方式命名。

2) 别名前加 as 使别名更容易识别。

正例：select t1.name from first_table as t1 , second_table as t2 where t1.id = t2.id;

使用了别名的多表查询：

```

1  select emp.name , dept.name
2  from tb_emp emp inner join tb_dept dept
3  on emp.dept_id = dept.id;

```

注意事项：

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。

1.3 外连接

外连接分为两种：左外连接 和 右外连接。

左外连接语法结构：

```
1  select  字段列表  from  表1 left  [ outer ] join 表2 on 连接条件
   ... ;
```

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。

右外连接语法结构：

```
1  select  字段列表  from  表1 right  [ outer ] join 表2 on 连接条件
   ... ;
```

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。

案例：查询员工表中所有员工的姓名， 和对应的部门名称

```
1  -- 左外连接：以left join关键字左边的表为主表，查询主表中所有数据，以及和主表
   匹配的右边表中的数据
2  select emp.name , dept.name
3  from tb_emp AS emp left join tb_dept AS dept
4       on emp.dept_id = dept.id;
```

emp.name	dept.name
纪晓芙	学工部
周芷若	学工部
丁敏君	学工部
赵敏	学工部
鹿杖客	咨询部
鹤笔翁	咨询部
方东白	咨询部
张三丰	教研部
俞莲舟	教研部
宋远桥	教研部
陈友谅	<null> 没有匹配到数据，填充为null

案例：查询部门表中所有部门的名称， 和对应的员工名称

```
1  -- 右外连接
2  select dept.name , emp.name
3  from tb_emp AS emp right join tb_dept AS dept
4       on emp.dept_id = dept.id;
```

dept.name	emp.name
人事部	<null> 没有匹配到的数据，填充为null
咨询部	方东白
咨询部	鹤笔翁
咨询部	鹿杖客
学工部	赵敏
学工部	丁敏君
学工部	周芷若
学工部	纪晓芙
学工部	小昭
就业部	<null>
教研部	宋远桥
教研部	俞莲舟

注意事项：

左外连接和右外连接是可以相互替换的，只需要调整连接查询时SQL语句中表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。

1.4 子查询

1.4.1 介绍

SQL语句中嵌套select语句，称为嵌套查询，又称子查询。

```
1  SELECT * FROM t1 WHERE column1 = ( SELECT column1 FROM t2
... );
```

子查询外部的语句可以是insert / update / delete / select 的任何一个，最常见的是select。

根据子查询结果的不同分为：

1. 标量子查询（子查询结果为单个值[一行一列]）
2. 列子查询（子查询结果为一列，但可以是多行）
3. 行子查询（子查询结果为一行，但可以是多列）
4. 表子查询（子查询结果为多行多列[相当于子查询结果是一张表]）

子查询可以书写的位置：

1. where之后

2. from之后
3. select之后

1.4.2 标量子查询

子查询返回的结果是单个值(数字、字符串、日期等)，最简单的形式，这种子查询称为标量子查询。

常用的操作符： = <> > >= < <=

案例1：查询"教研部"的所有员工信息

可以将需求分解为两步：

1. 查询 "教研部" 部门ID
2. 根据 "教研部" 部门ID, 查询员工信息

```
1  -- 1.查询"教研部"部门ID
2  select id from tb_dept where name = '教研部';    #查询结果：2
3  -- 2.根据"教研部"部门ID, 查询员工信息
4  select * from tb_emp where dept_id = 2;
5
6  -- 合并出上两条SQL语句
7  select * from tb_emp where dept_id = (select id from tb_dept where
    name = '教研部');
```

-- 合并出上两条SQL语句

```
select * from tb_emp where dept_id = (select id from tb_dept where name = '教研部');
```

子查询结果：一行一列

id	username	password	name	gender	image	job	entrydate	dept_id	creat
1	jinyong	123456	金庸		1 1.jpg	4	2000-01-01	2	2022-12
2	zhangwuji	123456	张无忌		1 2.jpg	2	2015-01-01	2	2022-12
3	yangxiao	123456	杨逍		1 3.jpg	2	2008-05-01	2	2022-12
4	weiyixiao	123456	韦一笑		1 4.jpg	2	2007-01-01	2	2022-12
5	changyuchun	123456	常遇春		1 5.jpg	2	2012-12-05	2	2022-12
6	zhangsanfeng	123456	张三丰		1 14.jpg	2	2002-08-01	2	2022-12
7	yulianzhou	123456	俞莲舟		1 15.jpg	2	2011-05-01	2	2022-12
8	songyuanqiao	123456	宋远桥		1 16.jpg	2	2007-01-01	2	2022-12

案例2：查询在 "方东白" 入职之后的员工信息

可以将需求分解为两步：

1. 查询 方东白 的入职日期
2. 查询 指定入职日期之后入职的员工信息

```
1  -- 1.查询"方东白"的入职日期
2  select entrydate from tb_emp where name = '方东白';      #查询结果:
                        2012-11-01
3  -- 2.查询指定入职日期之后入职的员工信息
4  select * from tb_emp where entrydate > '2012-11-01';
5
6  -- 合并以上两条SQL语句
7  select * from tb_emp where entrydate > (select entrydate from tb_emp
      where name = '方东白');
```

-- 合并以上两条SQL语句

```
select * from tb_emp where entrydate > (select entrydate from tb_emp where name = '方东白');
```

Output db04.tb_dept Result 24 合并出上两条SQL语句

6 rows

	id	username	password	name	gender	image	job	entrydate	dept_id	crea
1	2	zhangwuji	123456	张无忌	1	2.jpg	2	2015-01-01	2	2022-1
2	5	changyuchun	123456	常遇春	1	5.jpg	2	2012-12-05	2	2022-1
3	6	xiaozhao	123456	小昭	2	6.jpg	3	2013-09-05	1	2022-1
4	8	zhouzhiruo	123456	周芷若	2	8.jpg	1	2014-11-09	1	2022-1
5	10	zhaomin	123456	赵敏	2	10.jpg	1	2013-09-05	1	2022-1
6	17	chenyouliang	123456	陈友谅	1	17.jpg	<null>	2015-03-21	<null>	2022-1

1.4.3 列子查询

子查询返回的结果是一列(可以是多行)，这种子查询称为列子查询。

常用的操作符：

操作符	描述
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

案例：查询"教研部"和"咨询部"的所有员工信息

分解为以下两步：

1. 查询 "销售部" 和 "市场部" 的部门ID
2. 根据部门ID，查询员工信息

```

1  -- 1.查询"销售部"和"市场部"的部门ID
2  select id from tb_dept where name = '教研部' or name = '咨询部';    #查询结果：3,2
3  -- 2.根据部门ID，查询员工信息
4  select * from tb_emp where dept_id in (3,2);
5
6  -- 合并以上两条SQL语句
7  select * from tb_emp where dept_id in (select id from tb_dept where
    name = '教研部' or name = '咨询部');

```

-- 合并以上两条SQL语句

select * from tb_emp where dept_id in (select id from tb_dept where name = '教研部' or name = '咨询部');

子查询结果：一行多列

id	username	password	name	gender	image	job	entrydate	dept_id	create
1	1 jinyong	123456	金庸	1	1.jpg	4	2000-01-01	2	2022-1
2	2 zhangwuji	123456	张无忌	1	2.jpg	2	2015-01-01		2 2022-1
3	3 yangxiao	123456	杨逍	1	3.jpg	2	2008-05-01		2 2022-1
4	4 weiyixiao	123456	韦一笑	1	4.jpg	2	2007-01-01		2 2022-1
5	5 changyuchun	123456	常遇春	1	5.jpg	2	2012-12-05		2 2022-1
6	11 luzhangke	123456	鹿杖客	1	11.jpg	5	2007-02-01	3	2022-1
7	12 hebiweng	123456	鹤笔翁	1	12.jpg	5	2008-08-18	3	2022-1
8	13 fangdongbai	123456	方东白	1	13.jpg	5	2012-11-01	3	2022-1
9	14 zhangsanfeng	123456	张三丰	1	14.jpg	2	2002-08-01		2 2022-1
10	15 yulianzhou	123456	俞莲舟	1	15.jpg	2	2011-05-01		2 2022-1
11	16 songyuanqiao	123456	宋远桥	1	16.jpg	2	2007-01-01		2 2022-1

1.4.4 行子查询

子查询返回的结果是一行（可以是多列），这种子查询称为行子查询。

常用的操作符：=、<>、IN、NOT IN

案例：查询与“韦一笑”的入职日期及职位都相同的员工信息

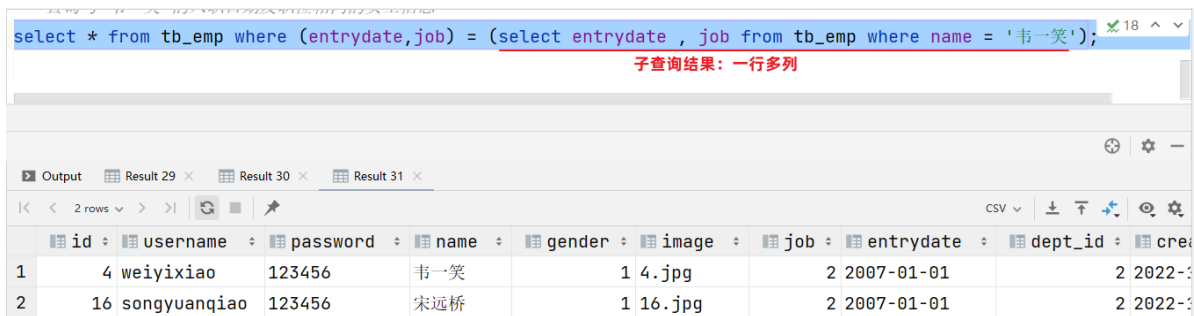
可以拆解为两步进行：

1. 查询“韦一笑”的入职日期及职位
2. 查询与“韦一笑”的入职日期及职位相同的员工信息

```

1  -- 查询"韦一笑"的入职日期 及 职位
2  select entrydate , job from tb_emp where name = '韦一笑'; #查询结果:
    2007-01-01 , 2
3  -- 查询与"韦一笑"的入职日期及职位相同的员工信息
4  select * from tb_emp where (entrydate,job) = ('2007-01-01',2);
5
6  -- 合并以上两条SQL语句
7  select * from tb_emp where (entrydate,job) = (select entrydate , job
    from tb_emp where name = '韦一笑');

```



	id	username	password	name	gender	image	job	entrydate	dept_id	create_time
1	4	weiyixiao	123456	韦一笑	1	4.jpg	2	2007-01-01	2	2022-01-01
2	16	songyuanqiao	123456	宋远桥	1	16.jpg	2	2007-01-01	2	2022-01-01

1.4.5 表子查询

子查询返回的结果是多行多列，常作为临时表，这种子查询称为表子查询。

案例：查询入职日期是 "2006-01-01" 之后的员工信息 ， 及其部门信息

分解为两步执行：

1. 查询入职日期是 "2006-01-01" 之后的员工信息
2. 基于查询到的员工信息，在查询对应的部门信息

```

1  select * from emp where entrydate > '2006-01-01';
2
3  select e.*, d.* from (select * from emp where entrydate > '2006-01-
    01') e left join dept d on e.dept_id = d.id ;

```

```
select e.*, d.*
from (select * from tb_emp where entrydate > '2006-01-01') AS e 左外查询以 e表为主表
left join tb_dept AS d
on e.dept_id = d.id ;
```

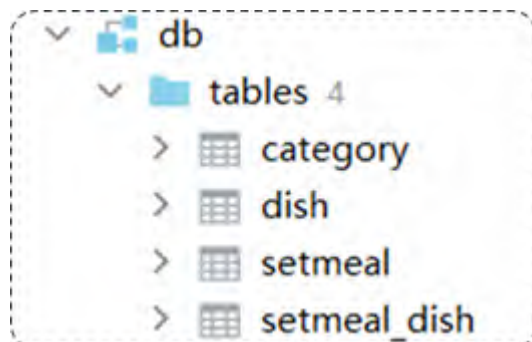
id	username	password	name	gender	image	job	entrydate	dept_id	city
6	8 zhouzhiruo	123456	周芷若	2	8.jpg	1	2014-11-09	1	2022
7	9 dingminjun	123456	丁敏君	2	9.jpg	1	2011-03-11	1	2022
8	10 zhaomin	123456	赵敏	2	10.jpg	1	2013-09-05	1	2022
9	11 luzhangke	123456	鹿杖客	1	11.jpg	5	2007-02-01	3	2022
10	12 hebiweng	123456	鹤笔翁	1	12.jpg	5	2008-08-18	3	2022
11	13 fangdongbai	123456	方东白	1	13.jpg	5	2012-11-01	3	2022
12	15 yulianzhou	123456	俞莲舟	1	15.jpg	2	2011-05-01	2	2022
13	16 songyuanqiao	123456	宋远桥	1	16.jpg	2	2007-01-01	2	2022
14	17 chenyouliang	123456	陈友谅	1	17.jpg	<null>	2015-03-21	没有匹配到, 填充null <null>	2022

1.5 案例

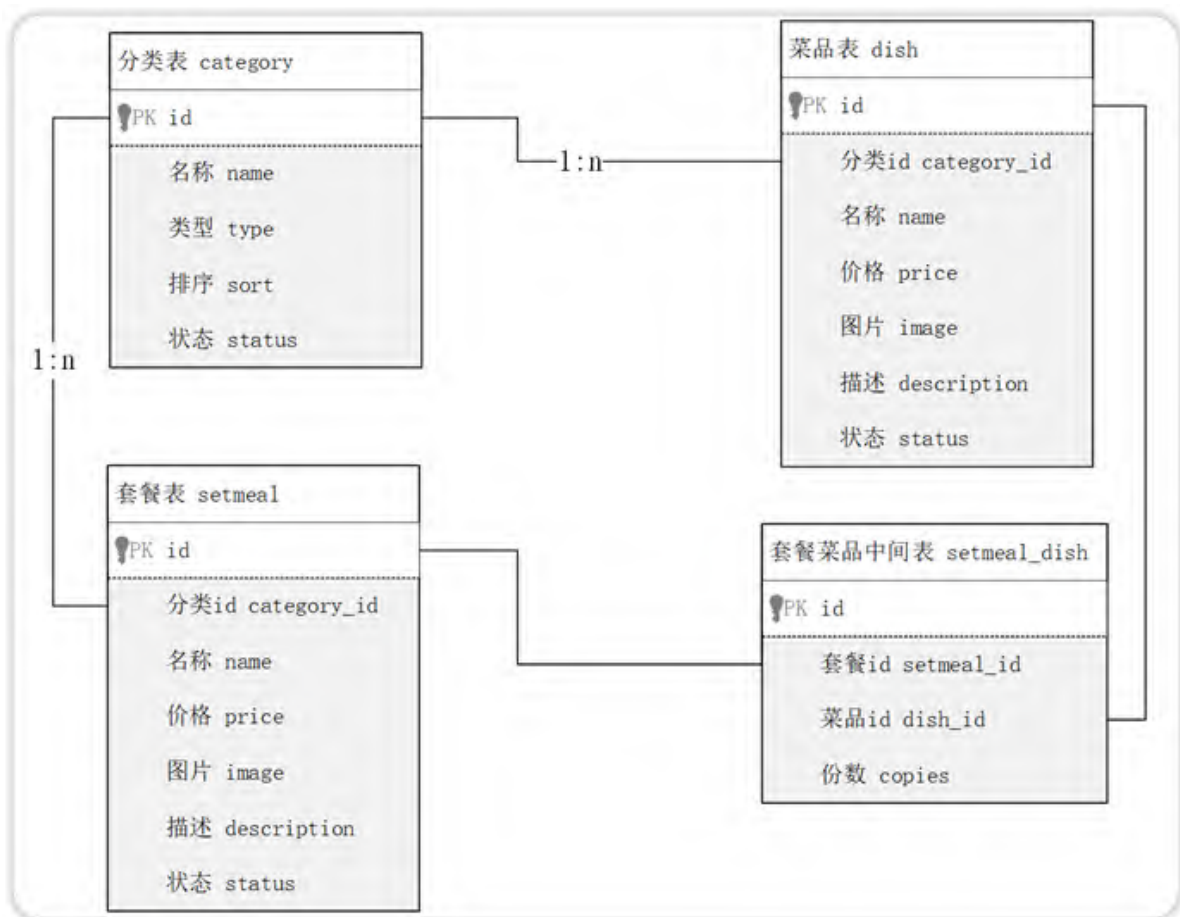
基于之前设计的多表案例的表结构，我们来完成今天的多表查询案例需求。

准备环境

将资料中准备好的多表查询的数据准备的SQL脚本导入数据库中。



- 分类表: category
- 菜品表: dish
- 套餐表: setmeal
- 套餐菜品关系表: setmeal_dish



需求实现

1. 查询价格低于 10元 的菜品的名称 、价格 及其 菜品的分类名称

```

1  /*查询技巧：
2      明确1：查询需要用到哪些字段
3          菜品名称、菜品价格 、 菜品分类名
4      明确2：查询的字段分别归属于哪张表
5          菜品表：[菜品名称、菜品价格]
6          分类表：[分类名]
7      明确3：如查多表，建立表与表之间的关联
8          菜品表.category_id = 分类表.id
9      其他：（其他条件、其他要求）
10         价格 < 10
11  */
12  select d.name , d.price , c.name
13  from dish AS d , category AS c
14  where d.category_id = c.id
15         and d.price < 10;
  
```


-- 查询价格低于 10 元 的菜品的名称、价格 及其 菜品的分类名称

```

select d.name , d.price , c.name
from dish AS d , category AS c
where d.category_id = c.id
and d.price <10;

```

	d.name	price	c.name
1	王老吉	6.00	酒水饮料
2	北冰洋	4.00	酒水饮料
3	雪花啤酒	4.00	酒水饮料
4	米饭	2.00	传统主食
5	馒头	1.00	传统主食
6	鸡蛋汤	4.00	汤类
7	平菇豆腐汤	6.00	汤类

2. 查询所有价格在 10元(含)到50元(含)之间 且 状态为"起售"的菜品名称、价格及其分类名称
(即使菜品没有分类 , 也要将菜品查询出来)

```

1 select d.name , d.price, c.name
2 from dish AS d left join category AS c on d.category_id = c.id
3 where d.price between 10 and 50
4 and d.status = 1;

```

```

select d.name , d.price, c.name
from dish AS d left join category AS c on d.category_id = c.id
where d.price between 10 and 50
and d.status = 1;

```

	d.name	price	c.name
1	蜀味水煮草鱼	38.00	经典川菜
2	清炒小油菜	18.00	新鲜时蔬
3	蒜蓉娃娃菜	18.00	新鲜时蔬
4	清炒西兰花	18.00	新鲜时蔬
5	炆炒圆白菜	18.00	新鲜时蔬

3. 查询每个分类下最贵的菜品, 展示出分类的名称、最贵的菜品的价格

```

1 select c.name , max(d.price)
2 from dish AS d , category AS c
3 where d.category_id = c.id
4 group by c.name;

```


select c.name , max(d.price)
from dish AS d , category AS c
where d.category_id = c.id
group by c.name; 按照分类名字进行分组

Output Result 45 Result 43 Result 44

5 rows

	name	max(d.price)
1	酒水饮料	6.00
2	传统主食	2.00
3	经典川菜	138.00
4	新鲜时蔬	18.00
5	汤类	6.00

4. 查询各个分类下 菜品状态为 "起售" , 并且 该分类下菜品总数量大于等于3 的 分类名称

```

1  /*查询技巧:
2      明确1: 查询需要用到哪些字段
3          分类名称、菜品总数量
4      明确2: 查询用到的字段分别归属于哪张表
5          分类表: [分类名]
6          菜品表: [菜品状态]
7      明确3: 如查多表, 建立表与表之间的关联
8          菜品表.category_id = 分类表.id
9      其他: (其他条件、其他要求)
10         条件: 菜品状态 = 1 (1表示起售)
11         分组: 分类名
12         分组后条件: 总数量 >= 3
13 */
14 select c.name , count(*)
15 from dish AS d , category AS c
16 where d.category_id = c.id
17         and d.status = 1 -- 起售状态
18 group by c.name -- 按照分类名分组
19 having count(*) >= 3; -- 各组后筛选菜品总数据>=3

```

-- 查询各个分类下 菜品状态为 "起售" , 并且 该分类下菜品总数量大于等于3 的 分类名称

select c.name , count(*)
from dish AS d , category AS c
where d.category_id = c.id
and d.status = 1 -- 起售状态
group by c.name -- 按照分类名分组
having count(*) >= 3; -- 各组后筛选菜品总数据>=3

Output Result 45 Result 46 Result 47

3 rows

	name	count(*)
1	酒水饮料	3
2	经典川菜	8
3	新鲜时蔬	4

5. 查询出 "商务套餐A" 中包含了哪些菜品 (展示出套餐名称、价格, 包含的菜品名称、价格、份数)

```
1 select s.name, s.price, d.name, d.price, sd.copies
2 from setmeal AS s , setmeal_dish AS sd , dish AS d
3 where s.id = sd.setmeal_id and sd.dish_id = d.id
4 and s.name='商务套餐A';
```

-- 查询出 "商务套餐A" 中包含了哪些菜品 (展示出套餐名称、价格, 包含的菜品名称、价格、份数)

```
select s.name, s.price, d.name, d.price, sd.copies
from setmeal AS s , setmeal_dish AS sd , dish AS d
where s.id = sd.setmeal_id and sd.dish_id = d.id
and s.name='商务套餐A';
```

	s.name	s.price	d.name	d.price	copies
1	商务套餐A	20.00	王老吉	6.00	1
2	商务套餐A	20.00	米饭	2.00	1
3	商务套餐A	20.00	清炒西兰花	18.00	1

6. 查询出低于菜品平均价格的菜品信息 (展示出菜品名称、菜品价格)

```
1 -- 1.计算菜品平均价格
2 select avg(price) from dish; -- 查询结果: 37.736842
3 -- 2.查询出低于菜品平均价格的菜品信息
4 select * from dish where price < 37.736842;
5
6 -- 合并以上两条SQL语句
7 select * from dish where price < (select avg(price) from dish);
```

-- 合并以上两条SQL语句

```
select * from dish where price < (select avg(price) from dish);
```

	id	name	category_id	price	image	description	status	creat
1	1	王老吉	1	6.00	https://reggie-itcast.oss-cn-beijing...		1	2022-06
2	2	北冰洋	1	4.00	https://reggie-itcast.oss-cn-beijing...	还是小时候的味道	1	2022-06
3	3	雪花啤酒	1	4.00	https://reggie-itcast.oss-cn-beijing...		1	2022-06
4	4	米饭	2	2.00	https://reggie-itcast.oss-cn-beijing...	精选五常大米	1	2022-06
5	5	馒头	2	1.00	https://reggie-itcast.oss-cn-beijing...	优质面粉	1	2022-06
6	9	清炒小油菜	6	18.00	https://reggie-itcast.oss-cn-beijing...	原料: 小油菜	1	2022-06
7	10	蒜蓉娃娃菜	6	18.00	https://reggie-itcast.oss-cn-beijing...	原料: 蒜, 娃娃菜	1	2022-06
8	11	清炒西兰花	6	18.00	https://reggie-itcast.oss-cn-beijing...	原料: 西兰花	1	2022-06
9	12	炆炒圆白菜	6	18.00	https://reggie-itcast.oss-cn-beijing...	原料: 圆白菜	1	2022-06
10	18	鸡蛋汤	7	4.00	https://reggie-itcast.oss-cn-beijing...	配料: 鸡蛋, 紫菜	1	2022-06
11	19	平菇豆腐汤	7	6.00	https://reggie-itcast.oss-cn-beijing...	配料: 豆腐, 平菇	1	2022-06

2. 事务

场景：学工部整个部门解散了，该部门及部门下的员工都需要删除了。

- 操作：

```
1  -- 删除学工部
2  delete from dept where id = 1;  -- 删除成功
3
4  -- 删除学工部的员工
5  delete from emp where dept_id = 1; -- 删除失败（操作过程中出现错误：
    造成删除没有成功）
```

- 问题：如果删除部门成功了，而删除该部门的员工时失败了，此时就造成了数据的不一致。

要解决上述的问题，就需要通过数据库中的事务来解决。

2.1 介绍

在实际的业务开发中，有些业务操作要多次访问数据库。一个业务要发送多条SQL语句给数据库执行。需要将多次访问数据库的操作视为一个整体来执行，要么所有的SQL语句全部执行成功。如果其中有一条SQL语句失败，就进行事务的回滚，所有的SQL语句全部执行失败。

简而言之：事务是一组操作的集合，它是一个不可分割的工作单位。事务会把所有的操作作为一个整体一起向系统提交或撤销操作请求，即这些操作要么同时成功，要么同时失败。

事务作用：保证在一个事务中多次操作数据库表中数据时，要么全都成功，要么全都失败。

2.2 操作

MySQL中有两种方式进行事务的操作：

1. 自动提交事务：即执行一条sql语句提交一次事务。（默认MySQL的事务是自动提交）
2. 手动提交事务：先开启，再提交

事务操作有关的SQL语句：

SQL语句	描述
<code>start transaction; / begin ;</code>	开启手动控制事务
<code>commit;</code>	提交事务
<code>rollback;</code>	回滚事务

手动提交事务使用步骤：

- 第1种情况：开启事务 => 执行SQL语句 => 成功 => 提交事务
- 第2种情况：开启事务 => 执行SQL语句 => 失败 => 回滚事务

使用事务控制删除部门和删除该部门下的员工的操作：

```
1  -- 开启事务
2  start transaction ;
3
4  -- 删除学工部
5  delete from tb_dept where id = 1;
6
7  -- 删除学工部的员工
8  delete from tb_emp where dept_id = 1;
```

- 上述的这组SQL语句，如果如果执行成功，则提交事务

```
1  -- 提交事务 （成功时执行）
2  commit ;
```

- 上述的这组SQL语句，如果如果执行失败，则回滚事务

```
1  -- 回滚事务 （出错时执行）
2  rollback ;
```

2.3 四大特性

面试题：事务有哪些特性？

- 原子性 (Atomicity)：事务是不可分割的最小单元，要么全部成功，要么全部失败。
- 一致性 (Consistency)：事务完成时，必须使所有的数据都保持一致状态。

- **隔离性 (Isolation)**：数据库系统提供的隔离机制，保证事务在不受外部并发操作影响的独立环境下运行。
- **持久性 (Durability)**：事务一旦提交或回滚，它对数据库中的数据的改变就是永久的。

事务的四大特性简称为：ACID

- **原子性 (Atomicity)**：原子性是指事务包装的一组sql是一个不可分割的工作单元，事务中的操作要么全部成功，要么全部失败。
- **一致性 (Consistency)**：一个事务完成之后数据都必须处于一致性状态。

如果事务成功的完成，那么数据库的所有变化将生效。

如果事务执行出现错误，那么数据库的所有变化将会被回滚（撤销），返回到原始状态。

- **隔离性 (Isolation)**：多个用户并发的访问数据库时，一个用户的事务不能被其他用户的事务干扰，多个并发的任务之间要相互隔离。

一个事务的成功或者失败对于其他的事务是没有影响。

- **持久性 (Durability)**：一个事务一旦被提交或回滚，它对数据库的改变将是永久性的，哪怕数据库发生异常，重启之后数据亦仍然存在。

3. 索引

3.1 介绍

索引(index)：是帮助数据库高效获取数据的数据结构。

- 简单来讲，就是使用索引可以提高查询的效率。

测试没有使用索引的查询：

```
select * from tb_sku where sn = '100000003145008';
```

Output db06.tb_sku 2 x

db06> select * from tb_sku where sn = '100000003145008' SQL执行时间: 24秒左右
[2022-11-09 11:54:02] 1 row retrieved starting from 1 in 24 s 437 ms (execution: 24 s 363 ms, fetching: 74 ms)

添加索引后查询:

```
1  -- 添加索引
2  create index idx_sku_sn on tb_sku (sn); #在添加索引时, 也需要消耗时间
3
4  -- 查询数据 (使用了索引)
5  select * from tb_sku where sn = '100000003145008';
```

```
select * from tb_sku where sn = '100000003145008';
```

Output db06.tb_sku x

db06> select * from tb_sku where sn = '100000003145008' 执行时间: 95毫秒
[2022-11-09 12:00:01] 1 row retrieved starting from 1 in 95 ms (execution: 12 ms, fetching: 83 ms)

优点:

1. 提高数据查询的效率, 降低数据库的IO成本。
2. 通过索引列对数据进行排序, 降低数据排序的成本, 降低CPU消耗。

缺点:

1. 索引会占用存储空间。
2. 索引大大提高了查询效率, 同时却也降低了insert、update、delete的效率。

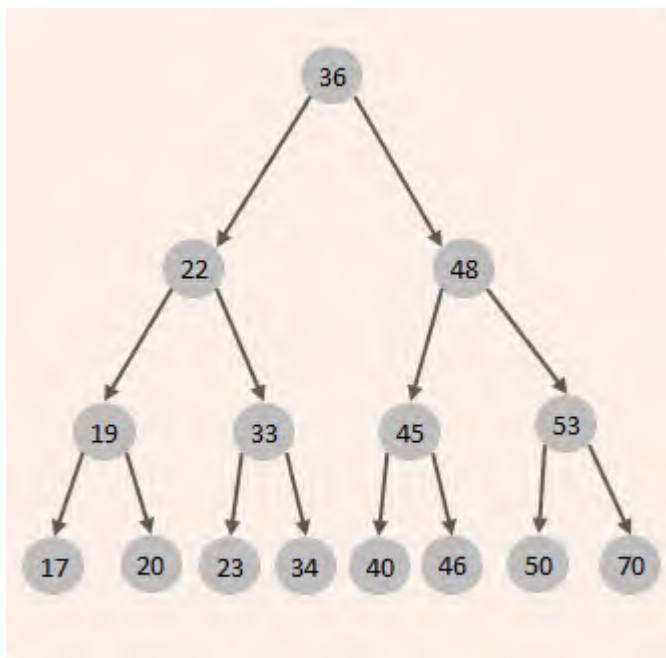
3.2 结构

MySQL数据库支持的索引结构有很多, 如: Hash索引、B+Tree索引、Full-Text索引等。

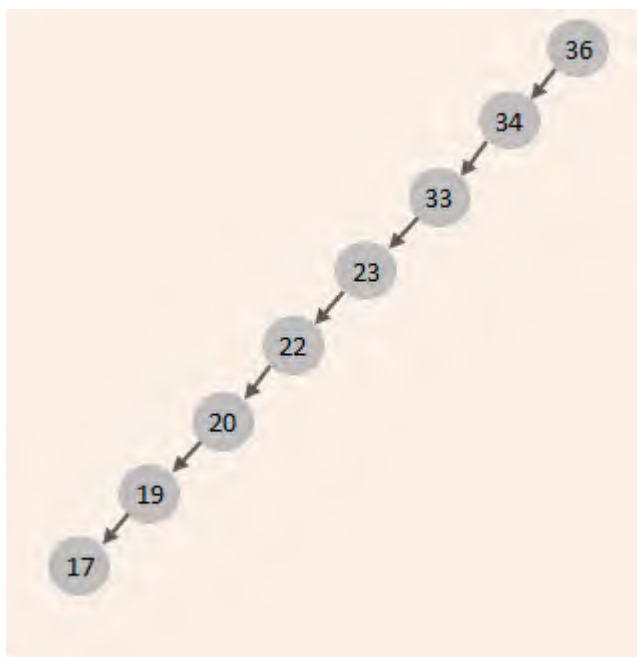
我们平常所说的索引, 如果没有特别指明, 都是指默认的 B+Tree 结构组织的索引。

在没有了解B+Tree结构前, 我们先回顾下之前所学习的树结构:

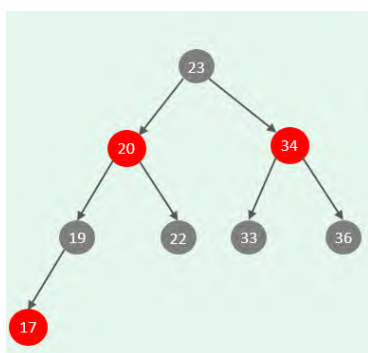
二叉查找树: 左边的子节点比父节点小, 右边的子节点比父节点大



当我们向二叉查找树保存数据时，是按照从大到小（或从小到大）的顺序保存的，此时就会形成一个单向链表，搜索性能会打折扣。



可以选择平衡二叉树或者是红黑树来解决上述问题。（红黑树也是一棵平衡的二叉树）



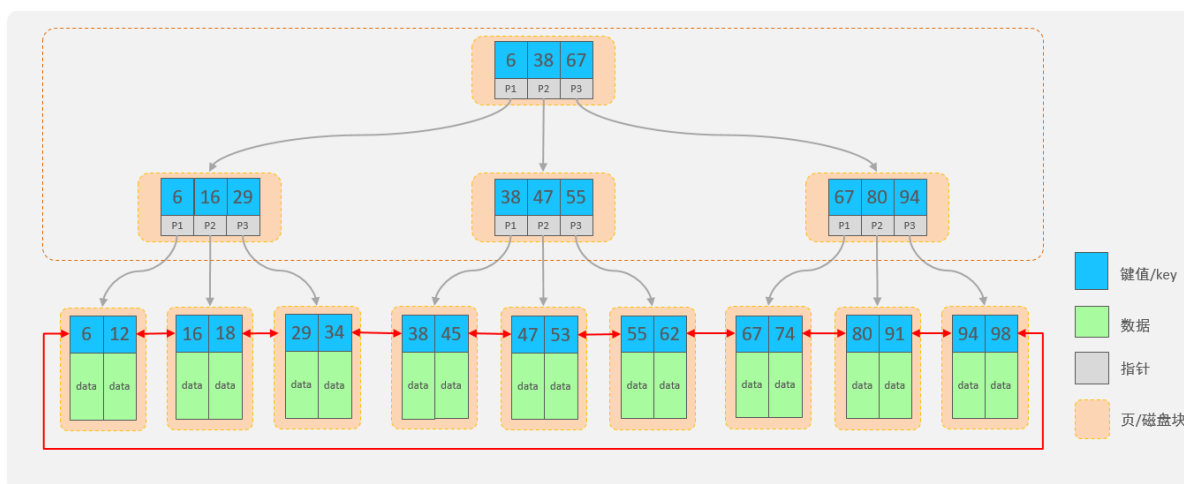
但是在MySQL数据库中并没有使用二叉搜索数或二叉平衡数或红黑树来作为索引的结构。

思考：采用二叉搜索树或者是红黑树来作为索引的结构有什么问题？

► 答案

说明：如果数据结构是红黑树，那么查询1000万条数据，根据计算树的高度大概是23左右，这样确实比之前的方式快了很多，但是如果高并发访问，那么一个用户有可能需要23次磁盘IO，那么100万用户，那么会造成效率极其低下。所以为了减少红黑树的高度，那么就得增加树的宽度，就是不再像红黑树一样每个节点只能保存一个数据，可以引入另外一种数据结构，一个节点可以保存多个数据，这样宽度就会增加从而降低树的高度。这种数据结构例如BTree就满足。

下面我们来看看B+Tree (多路平衡搜索树) 结构中如何避免这个问题：



B+Tree结构：

- 每一个节点，可以存储多个key (有n个key，就有n个指针)
- 节点分为：叶子节点、非叶子节点
 - 叶子节点，就是最后一层子节点，所有的数据都存储在叶子节点上
 - 非叶子节点，不是树结构最下面的节点，用于索引数据，存储的的是：key+指针
- 为了提高范围查询效率，叶子节点形成了一个双向链表，便于数据的排序及区间范围查询

拓展：

非叶子节点都是由key+指针域组成的，一个key占8字节，一个指针占6字节，而一个节点总共容量是16KB，那么可以计算出一个节点可以存储的元素个数： $16 \times 1024 \text{ 字节} / (8 + 6) = 1170$ 个元素。

- 查看mysql索引节点大小：`show global status like 'innodb_page_size';`
-- 节点大小：16384

当根节点中可以存储1170个元素，那么根据每个元素的地址值又会找到下面的子节点，每个子节点也会存储1170个元素，那么第二层即第二次IO的时候就会找到数据大概是：

$1170 \times 1170 = 135W$ 。也就是说B+Tree数据结构中只需要经历两次磁盘IO就可以找到135W条数据。

对于第二层每个元素有指针，那么会找到第三层，第三层由key+数据组成，假设key+数据总大小是1KB，而每个节点一共能存储16KB，所以一个第三层一个节点大概可以存储16个元素（即16条记录）。那么结合第二层每个元素通过指针域找到第三层的节点，第二层一共是135W个元素，那么第三层总元素大小就是： $135W \times 16$ 结果就是2000W+的元素个数。

结合上述分析B+Tree有如下优点：

- 千万条数据，B+Tree可以控制在小于等于3的高度
- 所有的数据都存储在叶子节点上，并且底层已经实现了按照索引进行排序，还可以支持范围查询，叶子节点是一个双向链表，支持从小到大或者从大到小查找

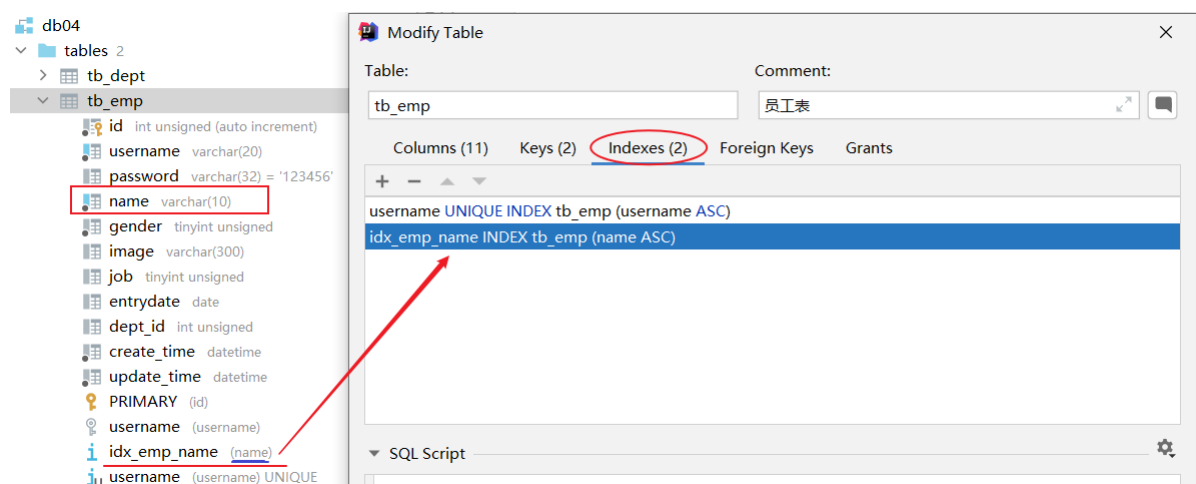
3.3 语法

创建索引

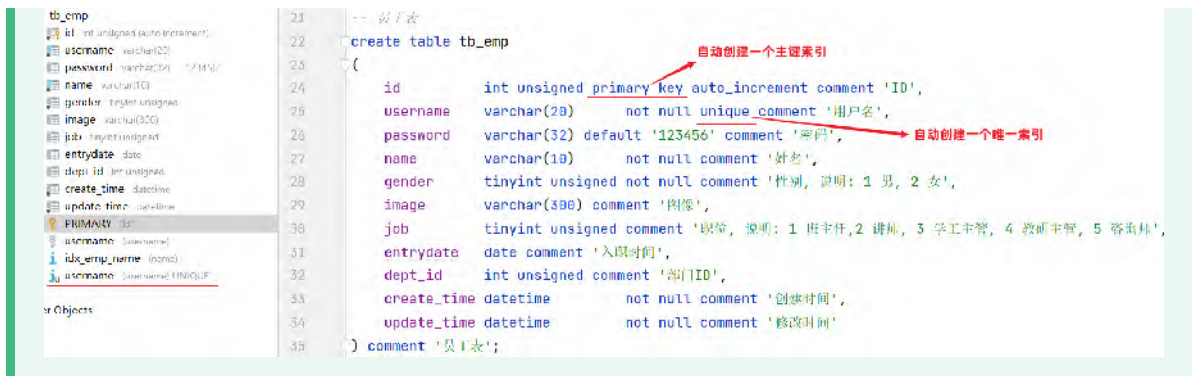
```
1 create [ unique ] index 索引名 on 表名 (字段名, ... ) ;
```

案例：为tb_emp表的name字段建立一个索引

```
1 create index idx_emp_name on tb_emp (name) ;
```



在创建表时，如果添加了主键和唯一约束，就会默认创建：主键索引、唯一约束



查看索引

```
1 show index from 表名;
```

案例：查询 tb_emp 表的索引信息

```
1 show index from tb_emp;
```

-- 查询：查询 tb_emp 表的索引信息

```
show index from tb_emp;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality
tb_emp	0	PRIMARY	1	id	A	17
tb_emp	0	username	1	username	A	17
tb_emp	1	idx_emp_name	1	name	A	17

删除索引

```
1 drop index 索引名 on 表名;
```

案例：删除 tb_emp 表中name字段的索引

```
1 drop index idx_emp_name on tb_emp;
```

注意事项：

- 主键字段，在建表时，会自动创建主键索引
- 添加唯一约束时，数据库实际上会添加唯一索引

