1 Lab5 SQL的复杂查询

在本章，我们会使用 Python 进行一些复杂的 SQL 查询，包括一些新的关键字，聚集函数，嵌套查询，两表的连接、窗口函数等等

1.1 一、常用关键字

首先我们需要创建表并且往其中插入相关的数据。

In [14]:
```
# 导入 Python 与 PostgreSQL 操作库
import psycopg2
conn = psycopg2.connect(host="172.16.253.154", port="5432", user="ecnu10215501412",
autocommit = psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT
conn.set_isolation_level(autocommit)
cur = conn.cursor()
```

创建四张表：

表一、S(SNO,SNAME,STATUS,CITY);

供应商表 S，供应商代码（SNO），供应商姓名（SNAME），供应 商状态（STATUS），供应商所在城市（CITY）

表二、P(PNO,PNAME,COLOR,WEIGHT);

零件表 P，零件代码（PNO），零件名字（PNAME），颜色 （COLOR），重量（WEIGHT）

表三、J(JNO,JNAME,CITY);

工程项目表 J，工程代码（JNO），工程名字（JNAME），工程所在城市（CITY）

表四、SPJ(SNO,PNO,JNO,QTY);

供应情况表 SPJ,供应商代码（SNO），零件代码（PNO），工程代码 （JNO），供应数量（QTY）

In [15]:
```
#如果这三张表已经存在，需要自己将这这些张表删除，参考 Lab4 的方法

drop_sql1 = "DROP TABLE IF EXISTS SPJ;"
drop_sql2 = "DROP TABLE IF EXISTS S;"
drop_sql3 = "DROP TABLE IF EXISTS P;"
drop_sql4 = "DROP TABLE IF EXISTS J;"

cur.execute(drop_sql1)
cur.execute(drop_sql2)
cur.execute(drop_sql3)
cur.execute(drop_sql4)

sql1="create table S \
(\
Sno char(2) unique,\
Sname char(6),\
Status char(2),\
City char(4),\
primary key(Sno)\
)"

sql2="create table P\
(\
Pno char(2) unique,\
Pname char(6),\
color char(2),\
weight int,\
primary key(Pno)\
);"

sql3="create table J\
(\
Jno char(2) unique,\
Jname char(8),\
CITY char(4),\
primary key(Jno)\
);"

sql4="create table SPJ\
(\
Sno char(2),\
Pno char(2),\
Jno char(2),\
QTY int,\
primary key(Sno,Pno,Jno),\
foreign key(Sno) references S(Sno),\
foreign key(Pno) references P(Pno),\
foreign key(Jno) references J(Jno)\
);"
cur.execute(sql1)
cur.execute(sql2)
cur.execute(sql3)
cur.execute(sql4)
```

In [16]:
```
#插入数据,确保只运行一次
sql1="insert into S(Sno,Sname,Status,City)\
values\
('S1','精益','20','天津'),\
('S2','盛锡','10','北京'),\
('S3','东方红','30','北京'),\
('S4','丰泰盛','20','天津'),\
('S5','为民','30','上海');"

sql2="insert into P(Pno,Pname,color,weight)\
values\
('P1','螺母','红',12),\
('P2','螺栓','绿',17),\
('P3','螺丝刀','蓝',14),\
('P4','螺丝刀','红',14),\
('P5','凸轮','蓝',40),\
('P6','齿轮','红',30);"

sql3="insert into J(Jno,Jname,CITY)\
values\
('J1','三建','北京'),\
('J2','一汽','长春'),\
('J3','弹簧厂','天津'),\
('J4','造船厂','天津'),\
('J5','机车厂','唐山'),\
('J6','无线电厂','常州'),\
('J7','半导体厂','南京');\
"

sql4="insert into SPJ(Sno,Pno,Jno,QTY)\
values\
('S1','P1','J1',200),\
('S1','P1','J3',100),\
('S1','P1','J4',700),\
('S1','P2','J2',100),\
('S2','P3','J1',400),\
('S2','P3','J2',200),\
('S2','P3','J4',500),\
('S2','P3','J5',400),\
('S2','P5','J1',400),\
('S2','P5','J2',100),\
('S3','P1','J1',200),\
('S3','P3','J1',200),\
('S4','P5','J1',100),\
('S4','P6','J3',300),\
('S4','P6','J4',200),\
('S5','P2','J4',100),\
('S5','P3','J1',200),\
('S5','P6','J2',200),\
('S5','P6','J4',500);"

cur.execute(sql1)
cur.execute(sql2)
cur.execute(sql3)
cur.execute(sql4)
```

In [4]:
```
#查看已有的数据表
cur.execute("""SELECT table_name FROM information_schema.tables
       WHERE table_schema = 'public'""")

for tuple in cur.fetchall():
    print(tuple)
```

```
('s',)
('spj',)
('p',)
('j',)
```

1.1.1 单表查询这种常用关键字

关键字	含义	用法
$ (not) like	相似匹配	常出现在where中，用来匹配字段中含有某一个特定模式的记录
$as	字段重命名	用来对select后面的字段重新命名
$order	排序	常用在一个sql查询的最后面，用来对查询结果做排序
$limit	限制输出数量	常用在一个sql查询的最后面，用来限制查询结果数量
$ (not) in	在（一个列表）之内	常出现在where中，用来判断字段值中是否在某个列表之内
$between . and .	在两数之间	常出现在where中，用来判断字段值中是否在两个值之间
$distinct	去重	用在select中，用于按照特定字段对结果去重
$case . when . then	多条件映射	常用在select中，用于将字段中满足条件的值映射成为一个新值

1.1.2 例1：查询零件表P中，零件名含有'螺'字的零件所有信息，并且按照零件重量降序排序

In [5]:
```
sql="select * from P \
where Pname like '%螺%' \
order by weight desc"
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
#like '%螺%':匹配含有‘螺’的
#like '%螺%':匹配以‘螺’开头的
#like '螺%':匹配以‘螺’结尾的
#desc：降序
#asc：升序
```

```
('P2', '螺栓    ', '绿', 17)
('P3', '螺丝刀   ', '蓝', 14)
('P4', '螺丝刀   ', '红', 14)
('P1', '螺母    ', '红', 12)
```

1.1.3 例2: 查询零件表P的所有信息，并且按照根据重量与25的关系来辨别轻重

In [6]:
```
sql="select *, \
case when weight<=25 then '轻' when weight>25 then '重' end as type \
from P \
order by weight desc"
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('P5', '凸轮    ', '蓝', 40, '重')
('P6', '齿轮    ', '红', 30, '重')
('P2', '螺栓    ', '绿', 17, '轻')
('P3', '螺丝刀   ', '蓝', 14, '轻')
('P4', '螺丝刀   ', '红', 14, '轻')
('P1', '螺母    ', '红', 12, '轻')
```

1.1.4 练习1：查询零工程项目表J中名字中含有‘厂’字的所有记录，并且需要根据城市名字来判断是不是直辖市

In [9]:
```
"""
('J3', '弹簧厂     ', '天津  ', '直辖市')
('J4', '造船厂     ', '天津  ', '直辖市')
('J5', '机车厂     ', '唐山  ', '非直辖市')
('J6', '无线电厂    ', '常州  ', '非直辖市')
('J7', '半导体厂    ', '南京  ', '非直辖市')
"""
sql="SELECT *  FROM J WHERE Jname LIKE '%厂%'"
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('J3', '弹簧厂     ', '天津  ')
('J4', '造船厂     ', '天津  ')
('J5', '机车厂     ', '唐山  ')
('J6', '无线电厂    ', '常州  ')
('J7', '半导体厂    ', '南京  ')
```

1.1.5 例3：查询零件表P中质量最轻的前三种零件的信息

In [8]:
```
sql="select * \
from P \
order by weight asc \
limit 3"
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('P1', '螺母    ', '红', 12)
('P3', '螺丝刀   ', '蓝', 14)
('P4', '螺丝刀   ', '红', 14)
```

1.2 二、聚集函数

类似于 MongoDB 中的聚集函数，PostgreSQL 也提供了常用的聚集函数，如 max, min, co
avg, sum 等。聚集函数的使用常常伴随着分组操作，即按照字段先分组，然后对不同分组的
定字段进行聚合操作。下面来看一个简单的聚合：

1.2.1 例4：零件表 P 中求不同颜色的零件的平均重量并且保留两位小数

In [9]:
```
sql="select color , round(avg(weight),2) as avg_color \
from P \
group by color"
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('红 ', Decimal('18.67'))
('绿 ', Decimal('17.00'))
('蓝 ', Decimal('27.00'))
```

如果不使用 group by 的话就是对所有表求均值；一般地，如果 select 出现了哪些除了聚集之外的字段，那么 group by 中也需要对应填上这些字段，不然可能会报错

以上，用到了 avg 聚合函数，和 round 这个 PostgreSQL 中内置的字段值转换函数，更多的函数可以参考https://blog.csdn.net/pg_hgdb/article/details/121612991 (https://blog.csdn.net/pg_hgdb/article/details/121612991)

In [10]:
```
# (1)
sql="select color as c, round(avg(weight),2) as avg_color \
from P \
where color !='绿' \
group by c"
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('红 ', Decimal('18.67'))
('蓝 ', Decimal('27.00'))
```

对于以上 SQL 查询，我们需要了解其查询的顺序：

1、先做 where 里面的筛选，即从 P 表中删除掉颜色为绿色的记录；

2、从留存的记录中抽取 select 中需要用到的列，这里是选出来了 color 和 weght 两列；

3、根据 group by 后面的字段进行分组，这里是按照 color 进行分组；

4、做聚集，化多为一，这里就是将不同颜色组中的重量聚合成他的平均数；

5、对聚集之后的字段做一些转换/重命名的操作

了解到 where 的筛选操作是发生在聚集之前的，那么如果我们想在聚集之后再对其做一些筛选需要怎么做呢？用到 having 字段！

having 字段和 where 字段都起到筛选作用，但是 having 发生在 where 之后，是用来对最终出结果做一个筛选的。

In [11]:
```
# (2)
sql="select color , round(avg(weight),2) as avg_color \
from P \
group by color \
having color !='绿'"
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('红 ', Decimal('18.67'))
('蓝 ', Decimal('27.00'))
```

以上两条查询分别使用了 where，having 实现了对绿色的剔除，请体会两者的区别

1.2.2 练习2：查询供应情况表 SPJ，查询不同供应商提供给工程项目（但不包括 J）的零件总数，并且筛选出总供应数大于600的记录

In [17]:
```
"""
('S1', 900)
('S5', 800)
('S2', 1200)
"""

sql="""
SELECT SPJ.SNO, SUM(SPJ.QTY) AS TotalQuantity
FROM SPJ
WHERE SPJ.JNO IN (SELECT JNO FROM J WHERE JNO != 'J')
GROUP BY SPJ.SNO
HAVING SUM(SPJ.QTY) > 600;
"""
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('S1', 1100)
('S5', 1000)
('S2', 2000)
```

1.3 三、嵌套查询

嵌套查询指的是两个查询互相嵌套，一个查询作为子查询出现在另一个查询中。子查询常出现在 select、from、where 这三处。

1.3.1 例5：查询当地没有供应商的工程项目全部信息

In [13]:
```
#子查询出现在 where 中
sql="select * \
from J \
where CITY not in \
(select distinct City from S) "
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('J2', '一汽      ', '长春  ')
('J5', '机车厂    ', '唐山  ')
('J6', '无线电厂  ', '常州  ')
('J7', '半导体厂  ', '南京  ')
```

1.3.2 例6：剔除绿色求均值

In [14]:
```
#子查询出现在 from 中
sql="select t.color,round(avg(t.weight),2) as avg_color \
from (select * from P where color !='绿')t \
group by t.color"
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('红 ', Decimal('18.67'))
('蓝 ', Decimal('27.00'))
```

1.3.3 例7：

In [15]:
```
#子查询出现在 select 中
sql="select \
(select round(avg(weight),2) as avg_weight from P) as avg_weight, \
(select max(QTY) as max_QTY from SPJ) as max_QTY"
#这种情况往往是用来输出一些独立的信息
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
(Decimal('21.17'), 700)
```

当然，如果觉得嵌套查询让一个查询过于繁琐且不直观，我们可以使用 with as 这个关键字来先创建一个临时表

1.3.4 例8：作用同例6

In [16]:
```
sql="with tmp as (select color,weight from P where color !='绿') \
select color,round(avg(weight),2) as avg_color \
from tmp \
group by color"
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('红 ', Decimal('18.67'))
('蓝 ', Decimal('27.00'))
```

1.3.5 练习3：从供应关系表 SPJ 中找到，供应商和项目工程都不在北京的记录并且按照供应数量升序排序。

提示：with as 创建多张临时表 with t as (select * from cx.over_test), t1 as (select * from t where ord='1')

In [6]:
```
"""
('S1', 'P1', 'J3', 100)
('S5', 'P2', 'J4', 100)
('S1', 'P2', 'J2', 100)
('S4', 'P6', 'J4', 200)
('S5', 'P6', 'J2', 200)
('S4', 'P6', 'J3', 300)
('S5', 'P6', 'J4', 500)
('S1', 'P1', 'J4', 700)
"""
sql="""
WITH
    SuppliersNotInBeijing AS (
        SELECT DISTINCT SNO
        FROM SPJ
        WHERE SNO NOT IN (SELECT SNO FROM S WHERE CITY = '北京')
    ),
    ProjectsNotInBeijing AS (
        SELECT DISTINCT JNO
        FROM SPJ
        WHERE JNO NOT IN (SELECT JNO FROM J WHERE CITY = '北京')
    )
SELECT SPJ.SNO, SPJ.PNO, SPJ.JNO, SPJ.QTY
FROM SPJ
WHERE SPJ.SNO IN (SELECT SNO FROM SuppliersNotInBeijing)
  AND SPJ.JNO IN (SELECT JNO FROM ProjectsNotInBeijing)
ORDER BY SPJ.QTY ASC;
"""

cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('S1', 'P1', 'J3', 100)
('S5', 'P2', 'J4', 100)
('S1', 'P2', 'J2', 100)
('S4', 'P6', 'J4', 200)
('S5', 'P6', 'J2', 200)
('S4', 'P6', 'J3', 300)
('S5', 'P6', 'J4', 500)
('S1', 'P1', 'J4', 700)
```

1.4 四、多表连接

多表连接查询出现在，我们需要的信息在往往分布在多张表中。来看一个多表查询的例子：

1.4.1 例9：查询供应关系表，并且需要将 Sno 转换成 Sname。

这个虽然通过 case--when 关键字可以实现，但是编写起来比较繁复而且不适用于大批量的转换。Sno 和 Sname 对应信息包含在 S 表中，因此我们需要通过连接两表来查询所需信息。

In [18]:
```
sql="select S.Sname,SPJ.Pno,SPJ.Jno,SPJ.QTY \
from SPJ,S \
where SPJ.Sno=S.sno"
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('精益    ', 'P1', 'J1', 200)
('精益    ', 'P1', 'J3', 100)
('精益    ', 'P1', 'J4', 700)
('精益    ', 'P2', 'J2', 100)
('盛锡    ', 'P3', 'J1', 400)
('盛锡    ', 'P3', 'J2', 200)
('盛锡    ', 'P3', 'J4', 500)
('盛锡    ', 'P3', 'J5', 400)
('盛锡    ', 'P5', 'J1', 400)
('盛锡    ', 'P5', 'J2', 100)
('东方红   ', 'P1', 'J1', 200)
('东方红   ', 'P3', 'J1', 200)
('丰泰盛   ', 'P5', 'J1', 100)
('丰泰盛   ', 'P6', 'J3', 300)
('丰泰盛   ', 'P6', 'J4', 200)
('为民    ', 'P2', 'J4', 100)
('为民    ', 'P3', 'J1', 200)
('为民    ', 'P6', 'J2', 200)
('为民    ', 'P6', 'J4', 500)
```

咱们继续来看看这个 SQL 的查询顺序：

1、首先就是执行 from 中的两张表的全连接，假设这两张表尺寸分别是 L1 行 M 列，L2 行 N 列，那么连接后的表尺寸是（L1 * L2）行（M + N）列。即表一的每一条记录都会拼接表2的每条记录出现在连接后的表中；

2、对连接后的表执行 where 筛选；

3、执行 select 等一些后续操作。

下面再来看一条查询，作用和上一条查询一样：

In [19]:
```
sql="select S.Sname,SPJ.Pno,SPJ.Jno,SPJ.QTY \
from SPJ full join S on SPJ.Sno=S.Sno"
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('精益    ', 'P1', 'J1', 200)
('精益    ', 'P1', 'J3', 100)
('精益    ', 'P1', 'J4', 700)
('精益    ', 'P2', 'J2', 100)
('盛锡    ', 'P3', 'J1', 400)
('盛锡    ', 'P3', 'J2', 200)
('盛锡    ', 'P3', 'J4', 500)
('盛锡    ', 'P3', 'J5', 400)
('盛锡    ', 'P5', 'J1', 400)
('盛锡    ', 'P5', 'J2', 100)
('东方红   ', 'P1', 'J1', 200)
('东方红   ', 'P3', 'J1', 200)
('丰泰盛   ', 'P5', 'J1', 100)
('丰泰盛   ', 'P6', 'J3', 300)
('丰泰盛   ', 'P6', 'J4', 200)
('为民    ', 'P2', 'J4', 100)
('为民    ', 'P3', 'J1', 200)
('为民    ', 'P6', 'J2', 200)
('为民    ', 'P6', 'J4', 500)
```

这里使用 on 这个关键字也实现和 where 一样的功能，但是这两者的区别是什么呢?

1.4.2 练习4：可百度，基于以上功能相同的两条语句，说明 where 和 on 的区别？哪一种方式效率高?

答:

ON子句通常用于连接操作，它定义了连接两个表时的匹配条件，在左连接中，ON子句控制哪些左表记录与右表记录匹配，即使没有匹配，右表结果列也会包含空值。相比之下，WHERE子句用于对连接后的结果集进行进一步筛选，它并不直接影响连接操作，而是用于在连接后的数据上应用筛选条件。

对于等值连接，两者性能差异不大。然而，在非等值连接中，ON子句通常更高效，因为它影响连接操作的匹配条件，有助于更好地利用索引和优化器的优化。

full join, inner join, left join 和 right join 的区别:

这是两张数据表连接的三种方式，都需要后面接上 on 关键字。

对与 `A inner join B on A.id=B.id` ： 只有 A，B 都含有的 id 记录才会出现在连接后的表中，保留了两表的共有字段;

对与 `A full join B on A.id=B.id` ： 任意一方含有的 id 记录都会出现在连接后的表中，不含 id 的表的相关字段会赋上空值，保留了两表的全部字段;

对与 `A left join B on A.id=B.id` ： 连接后的表的 id 仅来自于 A 表中，A 表不含的 id 但 B 表含有的 id 是不会出现在结果中的,即连接后的表包含了 A 表的全部字段。常用在对 A 表进行字段扩充的场景。

对与 A right join B on A.id=B.id ： 参考 left join, 这里 B 表是主角

更多介绍https://zhuanlan.zhihu.com/p/453311534 (https://zhuanlan.zhihu.com/p/453311534)

为了说明left join，我们先往S表插入一条SPJ表中所没有的Sno记录。注意以下sql由于主键约束，只能运行一次

In [20]:
```
sql1="insert into S \
values\
('S9','name','11','台湾')"
cur.execute(sql1)
```

1.4.3 例10：将 S 表和 SPJ 表通过 Sno 按照不同的方式连接

In [21]:
```
sql="select S.* from S"
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('S1', '精益  ', '20', '天津  ')
('S2', '盛锡  ', '10', '北京  ')
('S3', '东方红 ', '30', '北京  ')
('S4', '丰泰盛 ', '20', '天津  ')
('S5', '为民  ', '30', '上海  ')
('S9', 'name  ', '11', '台湾  ')
```

In [22]:
```
#left join
sql="select S.*,'||||',SPJ.* \
from S left join SPJ on SPJ.Sno=S.sno"
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('S1', '精益  ', '20', '天津  ', '||||', 'S1', 'P1', 'J1', 200)
('S1', '精益  ', '20', '天津  ', '||||', 'S1', 'P1', 'J3', 100)
('S1', '精益  ', '20', '天津  ', '||||', 'S1', 'P1', 'J4', 700)
('S1', '精益  ', '20', '天津  ', '||||', 'S1', 'P2', 'J2', 100)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P3', 'J1', 400)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P3', 'J2', 200)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P3', 'J4', 500)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P3', 'J5', 400)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P5', 'J1', 400)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P5', 'J2', 100)
('S3', '东方红 ', '30', '北京  ', '||||', 'S3', 'P1', 'J1', 200)
('S3', '东方红 ', '30', '北京  ', '||||', 'S3', 'P3', 'J1', 200)
('S4', '丰泰盛 ', '20', '天津  ', '||||', 'S4', 'P5', 'J1', 100)
('S4', '丰泰盛 ', '20', '天津  ', '||||', 'S4', 'P6', 'J3', 300)
('S4', '丰泰盛 ', '20', '天津  ', '||||', 'S4', 'P6', 'J4', 200)
('S5', '为民  ', '30', '上海  ', '||||', 'S5', 'P2', 'J4', 100)
('S5', '为民  ', '30', '上海  ', '||||', 'S5', 'P3', 'J1', 200)
('S5', '为民  ', '30', '上海  ', '||||', 'S5', 'P6', 'J2', 200)
('S5', '为民  ', '30', '上海  ', '||||', 'S5', 'P6', 'J4', 500)
('S9', 'name  ', '11', '台湾  ', '||||', None, None, None, None)
```

In [23]:
```
#full join
sql="select S.*,'||||',SPJ.* \
from S full join SPJ on SPJ.Sno=S.sno"
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('S1', '精益  ', '20', '天津  ', '||||', 'S1', 'P1', 'J1', 200)
('S1', '精益  ', '20', '天津  ', '||||', 'S1', 'P1', 'J3', 100)
('S1', '精益  ', '20', '天津  ', '||||', 'S1', 'P1', 'J4', 700)
('S1', '精益  ', '20', '天津  ', '||||', 'S1', 'P2', 'J2', 100)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P3', 'J1', 400)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P3', 'J2', 200)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P3', 'J4', 500)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P3', 'J5', 400)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P5', 'J1', 400)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P5', 'J2', 100)
('S3', '东方红 ', '30', '北京  ', '||||', 'S3', 'P1', 'J1', 200)
('S3', '东方红 ', '30', '北京  ', '||||', 'S3', 'P3', 'J1', 200)
('S4', '丰泰盛 ', '20', '天津  ', '||||', 'S4', 'P5', 'J1', 100)
('S4', '丰泰盛 ', '20', '天津  ', '||||', 'S4', 'P6', 'J3', 300)
('S4', '丰泰盛 ', '20', '天津  ', '||||', 'S4', 'P6', 'J4', 200)
('S5', '为民  ', '30', '上海  ', '||||', 'S5', 'P2', 'J4', 100)
('S5', '为民  ', '30', '上海  ', '||||', 'S5', 'P3', 'J1', 200)
('S5', '为民  ', '30', '上海  ', '||||', 'S5', 'P6', 'J2', 200)
('S5', '为民  ', '30', '上海  ', '||||', 'S5', 'P6', 'J4', 500)
('S9', 'name  ', '11', '台湾  ', '||||', None, None, None, None)
```

In [24]:
```
#inner join
sql="select S.*,'||||',SPJ.* \
from S inner join SPJ on SPJ.Sno=S.sno"
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('S1', '精益  ', '20', '天津  ', '||||', 'S1', 'P1', 'J1', 200)
('S1', '精益  ', '20', '天津  ', '||||', 'S1', 'P1', 'J3', 100)
('S1', '精益  ', '20', '天津  ', '||||', 'S1', 'P1', 'J4', 700)
('S1', '精益  ', '20', '天津  ', '||||', 'S1', 'P2', 'J2', 100)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P3', 'J1', 400)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P3', 'J2', 200)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P3', 'J4', 500)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P3', 'J5', 400)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P5', 'J1', 400)
('S2', '盛锡  ', '10', '北京  ', '||||', 'S2', 'P5', 'J2', 100)
('S3', '东方红 ', '30', '北京  ', '||||', 'S3', 'P1', 'J1', 200)
('S3', '东方红 ', '30', '北京  ', '||||', 'S3', 'P3', 'J1', 200)
('S4', '丰泰盛 ', '20', '天津  ', '||||', 'S4', 'P5', 'J1', 100)
('S4', '丰泰盛 ', '20', '天津  ', '||||', 'S4', 'P6', 'J3', 300)
('S4', '丰泰盛 ', '20', '天津  ', '||||', 'S4', 'P6', 'J4', 200)
('S5', '为民  ', '30', '上海  ', '||||', 'S5', 'P2', 'J4', 100)
('S5', '为民  ', '30', '上海  ', '||||', 'S5', 'P3', 'J1', 200)
('S5', '为民  ', '30', '上海  ', '||||', 'S5', 'P6', 'J2', 200)
('S5', '为民  ', '30', '上海  ', '||||', 'S5', 'P6', 'J4', 500)
```

观察上面 S9 的记录，体会这几种 join 方式的区别

1.4.4 练习5：将 SPJ 表中的 no 全部转化为 name

In [7]:
```
sql="""
SELECT S.SNAME AS Supplier, P.PNAME AS Part, J.JNAME AS Project, SPJ.QTY
FROM SPJ
JOIN S ON SPJ.SNO = S.SNO
JOIN P ON SPJ.PNO = P.PNO
JOIN J ON SPJ.JNO = J.JNO;
"""
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('精益    ', '螺母    ', '三建      ', 200)
('精益    ', '螺母    ', '弹簧厂    ', 100)
('精益    ', '螺母    ', '造船厂    ', 700)
('精益    ', '螺栓    ', '一汽      ', 100)
('盛锡    ', '螺丝刀  ', '三建      ', 400)
('盛锡    ', '螺丝刀  ', '一汽      ', 200)
('盛锡    ', '螺丝刀  ', '造船厂    ', 500)
('盛锡    ', '螺丝刀  ', '机车厂    ', 400)
('盛锡    ', '凸轮    ', '三建      ', 400)
('盛锡    ', '凸轮    ', '一汽      ', 100)
('东方红  ', '螺母    ', '三建      ', 200)
('东方红  ', '螺丝刀  ', '三建      ', 200)
('丰泰盛  ', '凸轮    ', '三建      ', 100)
('丰泰盛  ', '齿轮    ', '弹簧厂    ', 300)
('丰泰盛  ', '齿轮    ', '造船厂    ', 200)
('为民    ', '螺栓    ', '造船厂    ', 100)
('为民    ', '螺丝刀  ', '三建      ', 200)
('为民    ', '齿轮    ', '一汽      ', 200)
('为民    ', '齿轮    ', '造船厂    ', 500)
```

1.4.5 练习6：对零件表 P 按照 color 分组，并且求出每一组重量的方差。要求不使用方差或标准差的聚集函数，只使用 sum，avg，count 等上述提到的聚集函数。

In [8]:
```
"""
('红 ', 64.89)
('绿 ', 0)
('蓝 ', 169)
"""
sql="""
WITH GroupedP AS (
    SELECT P.COLOR, AVG(P.WEIGHT) AS AvgWeight, SUM(P.WEIGHT) AS SumWeight, COUNT(P.
    FROM P
    GROUP BY P.COLOR
)
SELECT P.COLOR,
    ROUND(SUM((P.WEIGHT - GroupedP.AvgWeight) * (P.WEIGHT - GroupedP.AvgWeight) / Gr
FROM P
JOIN GroupedP ON P.COLOR = GroupedP.COLOR
GROUP BY P.COLOR, GroupedP.AvgWeight, GroupedP.CountWeight;
"""
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('蓝 ', Decimal('169.00'))
('红 ', Decimal('64.89'))
('绿 ', Decimal('0.00'))
```

1.4.6 练习7：描述你所编写的练习6中 SQL 的查询顺序

答：首先进行数据的分组和准备，创建一个临时表 GroupedP存储按颜色分组的零件表P的平均重量、总重量和记录数，然后在主查询中进行数据的连接和方差计算，最终生成包括颜色和方差的结果集。

1.5 五、窗口函数

在例3，我们查询了 P 表中所有零件的前三重量。但是如果我们需要知道不同颜色分组前三的质量呢？那就需要用到窗口函数

1.5.1 例11：SPJ 表，查询不同供应商供应量前三的信息

In [26]:
```
sql="select *,rank() OVER (PARTITION BY Sno ORDER BY QTY DESC) as group_rank fro
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('S1', 'P1', 'J4', 700, 1)
('S1', 'P1', 'J1', 200, 2)
('S1', 'P2', 'J2', 100, 3)
('S1', 'P1', 'J3', 100, 3)
('S2', 'P3', 'J4', 500, 1)
('S2', 'P5', 'J1', 400, 2)
('S2', 'P3', 'J1', 400, 2)
('S2', 'P3', 'J5', 400, 2)
('S2', 'P3', 'J2', 200, 5)
('S2', 'P5', 'J2', 100, 6)
('S3', 'P3', 'J1', 200, 1)
('S3', 'P1', 'J1', 200, 1)
('S4', 'P6', 'J3', 300, 1)
('S4', 'P6', 'J4', 200, 2)
('S4', 'P5', 'J1', 100, 3)
('S5', 'P6', 'J4', 500, 1)
('S5', 'P3', 'J1', 200, 2)
('S5', 'P6', 'J2', 200, 2)
('S5', 'P2', 'J4', 100, 4)
```

从以上结果看出，除了 SPJ 表中的所有信息之外，结果中还多了一列，这就是窗口函数所起的作用---分组编号。

来看 rank() OVER (PARTITION BY Sno ORDER BY QTY DESC)：

rank()：代表了如何编号，这里是标上排名。当然也存在 max，avg ..等常用“编号”函数

PARTITION BY 某个字段：也就是按照这个分组，可省略那就是整张表作为一组。

ORDER BY 每个字段：查询的结果中每个分组是按照这个字段排序后展示的

值得注意的是，观看 S5 的分组：

rank() 使得窗口函数那一列的结果是：1-2-2-4

此外：

dense_rank() 可以输出：1-2-2-3

row_number() 可以输出：1-2-3-4

窗口函数在实际企业大规模的数据分析中是非常实用的一类技巧。如果考虑和数据分析或者数据开发相关的实习或工作的同学可以深入了解一下窗口函数，通过面试的可能性更大！！
https://blog.csdn.net/u011447403/article/details/122877396
(https://blog.csdn.net/u011447403/article/details/122877396)

1.5.2 练习8：对零件表按重量升序排序，附加一列累和

In [9]:
```
"""
每一列为当前所列的累和
('P1', '螺母    ', '红 ', 12, 12)
('P3', '螺丝刀   ', '蓝 ', 14, 40)
('P4', '螺丝刀   ', '红 ', 14, 40)
('P2', '螺栓    ', '绿 ', 17, 57)
('P6', '齿轮    ', '红 ', 30, 87)
('P5', '凸轮    ', '蓝 ', 40, 127)
"""
sql="""
SELECT PNO, PNAME, COLOR, WEIGHT,
       SUM(WEIGHT) OVER (ORDER BY WEIGHT ASC) AS CumulativeWeight
FROM P
ORDER BY WEIGHT ASC;
"""
cur.execute(sql)
for tuple in cur.fetchall():
    print(tuple)
```

```
('P1', '螺母    ', '红 ', 12, 12)
('P3', '螺丝刀   ', '蓝 ', 14, 40)
('P4', '螺丝刀   ', '红 ', 14, 40)
('P2', '螺栓    ', '绿 ', 17, 57)
('P6', '齿轮    ', '红 ', 30, 87)
('P5', '凸轮    ', '蓝 ', 40, 127)
```

In [28]:
```
#删除数据表
sql = "DROP TABLE SPJ; " + \
      "DROP TABLE S; " + \
      "DROP TABLE P; " + \
      "DROP TABLE J; "

cur.execute(sql)
```