数据库第三次作业

王晶 16340217

Task1

首先是完成 psycopg2 的安装，然后连接数据库

```
conn = psycopg2.connect(dbname="test", user="postgres",
        password="123456")

cur = conn.cursor()
```

然后创建名为 Sailors 的表，并插入数据，然后设置约束条件

```
#answer c
cur.execute(
        'CREATE TABLE sailors ('
        'sid integer,'
        'sname varchar(20),'
        'rating integer,'
        'age real'
        ')'
    )


cur.execute(
        "INSERT INTO sailors "
        "VALUES(22, 'Dustin', 7, 45.0),"
        "(29, 'Brutus', 1, 33.0),"
        "(31, 'Lubber', 8, 55.5),"
        "(32, 'Andy', 8, 25.5),"
        "(64, 'Horatio', 7, 35.0),"
        "(71, 'Zorba', 10, 16.0)"
    )

print 'Create Sailors!'

#answer e
cur.execute(
    'alter table sailors add primary key(sid)'
    )

cur.execute(
    'alter table sailors alter COLUMN sname set not null'
    )

print 'Set the constraint!'
```

输出结果如下：

```
Create Sailors!
Set the constraint!
```

然后是第 f 小题：

```
cur.execute(
    'update sailors set rating = rating+1 where rating < 7'
    )
```

执行后的结果

```
 sid |  sname   | rating | age
-----+----------+--------+------
  22 | Dustin   |      7 |   45
  31 | Lubber   |      8 | 55.5
  32 | Andy     |      8 | 25.5
  64 | Horatio  |      7 |   35
  71 | Zorba    |     10 |   16
  29 | Brutus   |      2 |   33
```

然后是创建名为 Boats 和 Reserves 的表，并且设置对应的约束条件：

Boats：

```
    cur.execute(
            'CREATE TABLE Boats ('
            'bid integer,'
            'bname varchar(20),'
            'color varchar(20),'
            'primary key(bid)'
            ')'
        )
```

Reserves：

```
    #answer h
    cur.execute(
            'CREATE TABLE Reserves ('
            'sid integer references sailors(sid), '
            'bid integer references Boats(bid), '
            'day DATE'
            ')'
        )
```

然后是第 i 小题：

为了在逻辑上更严密，添加了 distinct，虽然本题条件下，是否添加没什么影响

结果为：

```
Question i:

sname =  Andy

sname =  Brutus
```

第 j 小题：

创建索引：

```
#answer j
cur.execute('create index idx_btree_day on Reserves using btree(day)')

print 'Create Btree Index!'
```

然后进行操作，得到结果：

```
Question j:

sname =  Zorba

sname =  Brutus
```

第 k 小题：

一开始想到直接选择非红色的 Reserves，后来发现，存在一种情况就是，有人即订过红色，也订过非红色，这样的话就会出现错误，所以要选出订过其他颜色的并除去订过红色的。

输出结果：

```
Question k:

sid =  64
```

第 l 题：

在我看来，用 python 或其他语言来连接数据库，就能通过所写的程序来直接获取数据库中的内容，并且可以直接使用了，而不是靠人力来手动从数据库中获取数据，再复制粘贴到程序中，然后再使用。并且也省去了文件读写操作的步骤，可以将内容寄放在数据库中。

Task2

首先是创建表 families_j，然后插入数据：

```
cur.execute(
        'CREATE TABLE families_j('
        'id integer, '
        'profile json'
        ')'
    )


cur.execute(
"""INSERT INTO families_j
VALUES(1,
'{"name":"Gomez","members":[
    {"member":{"relation":"padre", "name":"Alex"}},
    {"member":{"relation":"madre", "name":"Sonia" }},
    {"member":{"relation":"hijo", "name":"Brandon"}},
    {"member":{"relation":"hija", "name":"Azaleah"}}
    ]}')"""
)
```

插入成功：

```
Create families_j!
```

然后是第 c 小题：

结果如下：

```
Question c:

number =  4

Alex
Sonia
Brandon
Azaleah
```

然后将这些名字转换为 JSON 格式数据，插入新创建的表 families_b 中，可以使用 python 自带的 json 包：

```
tempname = []

for row in rows[0][0]:
    tempname.append(row['member']['name'])
    print row['member']['name']

print '\n'


tempjs = {
    "num":len(rows[0][0]),
    "name":tempname
}

templs = json.dumps(tempjs)

print templs, '\n'

cur.execute(
        'CREATE TABLE families_b ('
        'id integer, '
        'profile json'
        ')'
    )


cur.execute(
        "INSERT INTO families_b "
        "VALUES(1, '%s'"%(templs) + ")"
    )
```

然后创建结果：

```
{"num": 4, "name": ["Alex", "Sonia", "Brandon", "Azaleah"]}
```

最后是第 e 小题：

表 families_j：

```
families_j:
1
{u'name': u'Gomez', u'members': [{u'member': {u'relation': u'padre', u'name': u'Alex'}}, {
u'member': {u'relation': u'madre', u'name': u'Sonia'}}, {u'member': {u'relation': u'hijo',
 u'name': u'Brandon'}}, {u'member': {u'relation': u'hija', u'name': u'Azaleah'}}]}
```

表 families_b：

```
families_b:
1
{u'num': 4, u'name': [u'Alex', u'Sonia', u'Brandon', u'Azaleah']}
```

因为使用 python2.7 的原因，所以字符串输出 unicode 格式时会带 u

那么直接在 postgresql 中看的话：

```
test=# select * from families_j;
 id |                          profile
----+-----------------------------------------------------------
  1 | {"name":"Gomez","members":[                                +
    |         {"member":{"relation":"padre", "name":"Alex"}},    +
    |         {"member":{"relation":"madre", "name":"Sonia" }},+
    |         {"member":{"relation":"hijo", "name":"Brandon"}},+
    |         {"member":{"relation":"hija", "name":"Azaleah"}} +
    |         ]}
(1 row)
```

和

```
test=# select * from families_b;
 id |                          profile
----+-----------------------------------------------------------
  1 | {"num": 4, "name": ["Alex", "Sonia", "Brandon", "Azaleah"]}
```

区别：

区别在于，families_j 中的 json 数据有 family name，还有存有家庭成员名字和关系的数组，而 families_b 中则有成员数，还有各个家庭成员名字的数组。