

# Insert Assignment Title Here

## 02807 Computational Tools for Big Data

Anonymous authors

Insert hand in date here

## 1 Exercide 5

### 1.1 Exercise 5.1

#### SQLite:

We run the following python code in order to find all tables in the database:

```
self.conn = sqlite3.connect(os.path.dirname(__file__) + '/../data/northwind.db')
self.conn.text_factory = str
c = self.conn.cursor()
tables = c.execute("SELECT * FROM sqlite_master WHERE type='table';").fetchall()
self.conn.close()
```

Printing *tables* yields the following result:

```
['Categories',
 'sqlite_sequence',
 'CustomerCustomerDemo',
 'CustomerDemographics',
 'Customers',
 'Employees',
 'EmployeeTerritories',
 'Order Details',
 'Orders',
 'Products',
 'Regions',
 'Shippers',
 'Suppliers',
 'Territories']
```

#### MongoDB:

We run the following python code in order to find all customers who reside in the city of Berlin.

```
self.client = MongoClient()
self.db = self.client['Northwind']
documents = []
for doc in self.db['customers'].find({'City': 'Berlin'}):
    documents += [doc]
```

If we print *documents* we get the following result.

```
[{'Address': u'Obere Str. 57',
  'City': u'Berlin',
  'CompanyName': u'Alfreds Futterkiste',
  'ContactName': u'Maria Anders',
  'ContactTitle': u'Sales Representative',
  'Country': u'Germany',
  'CustomerID': u'ALFKI',
  'Fax': u'030-0076545',
  'Phone': u'030-0074321',
  'PostalCode': 12209,
```

```
u'Region': u'NULL',
u'_id': ObjectId('5626242620dcd23954bcb018')}]
```

## 1.2 Exercise 5.2

In order to help with using MongoDB code in exercise 5 we have implemented the following helper class:

```
class Mongodb(object):
    def __init__(self, dbname):
        self.client = MongoClient()
        self.db = self.client[dbname]

    def group_by(self, obj, key, condition, reduce_function, initial, finalize=None):
        documents = []
        for doc in self.db[obj].group(key=key, condition=condition, reduce=reduce_function, ↵
            initial=initial, finalize=finalize):
            documents += [doc]
        return documents

    def find_all(self, obj):
        documents = []
        for doc in self.db[obj].find():
            documents += [doc]
        return documents

    def find_by(self, obj, selector):
        documents = []
        for doc in self.db[obj].find(selector):
            documents += [doc]
        return documents

    def join(self, collection1, obj, fk1, fk2):
        return filter(
            lambda x: True if x[fk1] == obj[fk2] else False,
            collection1
        )

    def join_in(self, collection1, collection2, fk):
        return filter(
            lambda x: True if x[fk] in collection2 else False,
            collection1
        )
```

The actual code we use to solve exercise 5.2 looks as follows:

```
db = Mongodb('Northwind')

orders = db.find_by('orders', {'CustomerID': 'ALFKI'})

order_details = db.find_by(
    'order-details', {
        'OrderID': {
            '$in': map(lambda x: x['OrderID'], orders)
        }
    }
)

products = db.find_by(
    'products', {
        'ProductID': {
            '$in': map(lambda x: x['ProductID'], order_details)
        }
    }
)

# exercise 5.2
def exercise5_2():
    for order in orders:
        print 'Order ID:' + str(order['OrderID'])
        pprint(
            map(
                lambda y: y['ProductName'],
                db.join_in(
                    products,
                    map(
                        lambda x: x['ProductID'],
```

```

        db.join(order_details , order , 'OrderID ' , 'OrderID ' )
    ),
    'ProductID '
)
)
)

```

If we run the function `exercise5_2` we get the following output:

```
Order ID:10643
[u'R\xf6ssle Sauerkraut', u'Chartreuse verte', u'Spegesild']
Order ID:10692
[u'Vegie-spread']
Order ID:10702
[u'Aniseed Syrup', u'Lakkalik\xf6ri']
Order ID:10835
[u'Raclette Courdavault', u'Original Frankfurter gr\xfcne So\xdf']
Order ID:10952
[u"Grandma's Boysenberry Spread", u'R\xf6ssle Sauerkraut']
Order ID:11011
[u'Escargots de Bourgogne', u'Flotemysost']
```

### 1.3 Exercise 5.3

Given the same helper class we use the following function to solve exercise 5.3:

```
def exercise5_3():
    for order in orders:
        order_products = map(
            lambda y: y['ProductName'],
            db.join_in(
                products,
                map(
                    lambda x: x['ProductID'],
                    db.join(order_details, order, 'OrderID', 'OrderID')
                ),
                'ProductID'
            )
        )

    if len(order_products) > 1:
        print 'Order ID:' + str(order['OrderID'])
        pprint(order_products)
```

Running the function `exercise53` yields the following result:

```
Order ID:10643
[u'R\x{f6}ssle Sauerkraut', u'Chartreuse verte', u'Spegesild']
Order ID:10702
[u'Aniseed Syrup', u'Lakkalik\x{f6}\x{f6}ri']
Order ID:10835
[u'Raclette Courdavault', u'Original Frankfurter gr\x{fc}ne So\x{df}e']
Order ID:10952
[u"Grandma's Boysenberry Spread", u'R\x{f6}ssle Sauerkraut']
Order ID:11011
[u'Escargots de Bourgogne', u'Flotemysost']
```

### 1.4 Exercise 5.4

Given the same prerequisite helper class, we use the following function to solve exercise 5.4:

```
def exercise5_4():
    order_details = db.find_by('order-details', {'ProductID': 7})

    orders = db.find_by('orders', {
        'OrderID': {
            '$in': map(lambda x: x['OrderID'], order_details)
        }
    })
```

```

customers = db.find_by('customers', {
    'CustomerID': {
        '$in': map(lambda x: x['CustomerID'], orders)
    }
})

return customers

```

Running the function *exercise5<sub>4</sub>* and printing the result yields the following:

```

Total number of customers: 20
[u'Laurence Lebihan',
 u'Elizabeth Lincoln',
 u'Victoria Ashworth',
 u'Roland Mendel',
 u'Martine Ranc\xe9',
 u'Maria Larsson',
 u'Andr\xe9 Fonseca',
 u'Daniel Tonini',
 u'Carlos Gonz\xe1lez',
 u'Yvonne Moncada',
 u'Ann Devon',
 u'Henriette Pfalzheim',
 u'Horst Kloss',
 u'Paula Wilson',
 u'Maurizio Moroni',
 u'Jonas Bergulfen',
 u'Jose Pavarotti',
 u'Art Braunschweiger',
 u'Palle Ibsen',
 u'Mary Saveley']

```

## 1.5 Exercise 5.5

In order to solve exercise 5.5 we reuse the function from exercise 5.4:

```

def exercise5_5():
    customers = exercise5_4()

    orders = db.find_by('orders', {
        'CustomerID': {
            '$in': map(lambda x: x['CustomerID'], customers)
        }
    })

    order_details = db.find_by('order-details', {
        'OrderID': {
            '$in': map(lambda x: x['OrderID'], orders)
        }
    })

    products = db.find_by('products', {
        'ProductID': {
            '$in': map(lambda x: x['ProductID'], order_details),
            '$not': re.compile("7")
        }
    })

    pprint(len(products))
    pprint(map(lambda x: x['ProductName'], products))

```

Running the function *exercise5<sub>5</sub>* yields the following result:

```

Product count: 76
[u'Chai',
 u'Aniseed Syrup',
 u"Chef Anton's Cajun Seasoning",
 u'Chef Anton's Gumbo Mix',
 u"Grandma's Boysenberry Spread",
 u'Chang',
 u"Uncle Bob's Organic Dried Pears",
 u'Mishi Kobe Niku',
 u'Ikura',
 u'Northwoods Cranberry Sauce',
 u'Queso Cabrales',
 u'Queso Manchego La Pastora',

```

```

u 'Konbu',
u 'Tofu',
u 'Genen Shouyu',
u 'Pavlova',
u 'Alice Mutton',
u 'Carnarvon Tigers',
u 'Teatime Chocolate Biscuits',
u "Sir Rodney's Marmalade",
u "Sir Rodney's Scones",
u "Gustaf's Kn\xe4ckebr\xf6d",
u 'Tunnbr\xf6d',
u 'Guaran\xel Fant\xelstica',
u 'Gumb\xe4r Gummib\xe4rchen',
u 'NuNuCa Nu\xdf-Nougat-Creme',
u 'Schoggi Schokolade',
u 'R\xf6ssle Sauerkraut',
u 'Th\xfcringer Rostbratwurst',
u 'Nord-Ost Matjeshering',
u 'Gorgonzola Telino',
u 'Mascarpone Fabioli',
u 'Geitost',
u 'Sasquatch Ale',
u 'Steeleye Stout',
u 'Inlagd Sill',
u 'Gravad lax',
u 'C\xf4te de Blaye',
u 'Chartreuse verte',
u 'Boston Crab Meat',
u "Jack's New England Clam Chowder",
u 'Singaporean Hokkien Fried Mee',
u 'Ipoh Coffee',
u 'Gula Malacca',
u 'Rogede sild',
u 'Spegesild',
u 'Zaanse koeken',
u 'Chocolade',
u 'Maxilaku',
u 'Valkoinen suklaa',
u 'Manjimup Dried Apples',
u 'Filo Mix',
u 'Perth Pasties',
u 'Tourti\xe8re',
u 'P\xe2t\xe9 chinois',
u 'Gnocchi di nonna Alice',
u 'Ravioli Angelo',
u 'Escargots de Bourgogne',
u 'Raclette Courdavault',
u 'Camembert Pierrot',
u "Sirop d'érable",
u 'Tarte au sucre',
u 'Vegie-spread',
u 'Wimmers gute Semmelkn\xf6del',
u 'Louisiana Fiery Hot Pepper Sauce',
u 'Louisiana Hot Spiced Okra',
u 'Scottish Longbreads',
u 'Gudbrandsdalsost',
u 'Outback Lager',
u 'Flotemysost',
u 'Mozzarella di Giovanni',
u 'R\xf6d Kaviar',
u 'Longlife Tofu',
u 'Rh\xf6n-Klosterbier',
u 'Lakkalik\xf6ri',
u 'Original Frankfurter gr\xfcne So\xdf']

```

## 1.6 Exercise 5.6

To solve exercise 5.6 we use the following function:

```

def exercise5_6():
    customers = exercise5_4()

    orders = db.find_by('orders', {
        'CustomerID': {
            '$in': map(lambda x: x['CustomerID'], customers)
        }
    })

    gb = db.group_by(
        'order-details',
        key={'ProductID': 1},

```

```

        condition={
            'OrderID': {
                '$in': map(lambda x: x['OrderID'], orders)
            }
        },
        reduce_function='function (current, result) { result.total += current.Quantity; }',
        initial={'total': 0}
    )

    gb.sort(key=lambda x: x['total'], reverse=True)

    pprint(gb[0])

    product = db.find_by('products', {
        'ProductID': gb[0]['ProductID']
    })

    pprint(product)

```

Running the function *exercise56* yields the following result:

```


{u'ProductID': 60.0, u'total': 769.0}
[{u'CategoryID': 4,
  u'Discontinued': 0,
  u'ProductID': 60,
  u'ProductName': u'Camembert Pierrot',
  u'QuantityPerUnit': u'15 - 300 g rounds',
  u'ReorderLevel': 0,
  u'SupplierID': 28,
  u'UnitPrice': 34.0,
  u'UnitsInStock': 19,
  u'UnitsOnOrder': 0,
  u'_id': ObjectId('5626242620dcd23954bcb88')}]

```

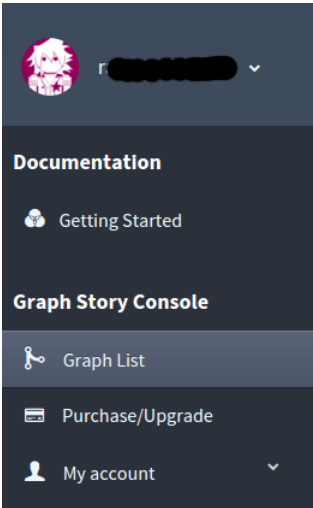
## 2 Exercise 6

### 2.1 Exercise 6.1

Here you can see we have created a Graph Story account and generated a database:



Tools-Database	
URL	https://neo-561[redacted].do-stories.graphstory.com:7473/browser/
Host	neo-561[redacted].do-stories.graphstory.com:7473
Port	7473
Username	561[redacted]
Password	fBG[redacted]



Then we removed all data from the database:

```
$ MATCH (n)-[r]-() WHERE ID(n)>=0 DELETE n;
```

Graph

(no changes, no rows)

Rows

Returned 0 rows in 3709 ms.

Lastly, we loaded all the data into the database, using the given code. I've chosen to only include the output of the last line of code, which should be enough proof that we have done them all.

```
$ LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/order-details.csv" AS ro
```

Graph

Set 12930 properties, created 2155 relationships, statement executed in 2109 ms.

Rows

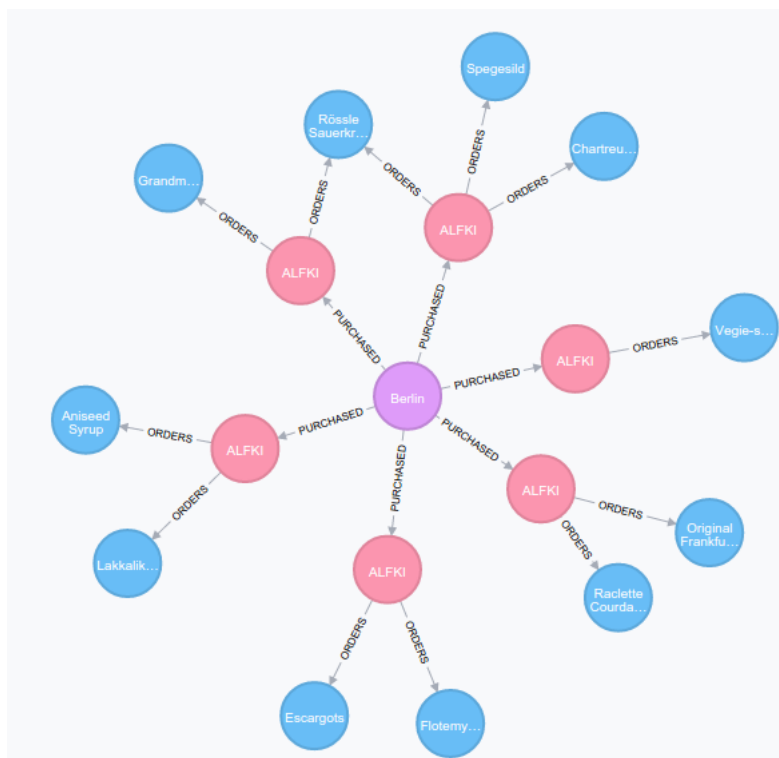
Returned 0 rows.

## 2.2 Exercise 6.2

Using the following query, we are able to find all orders for the customer with customerID 'ALFKI', including the products in those orders.

```
MATCH (customer {customerID: 'ALFKI'})---(orders) RETURN orders
```

This returns the following graph from the database:



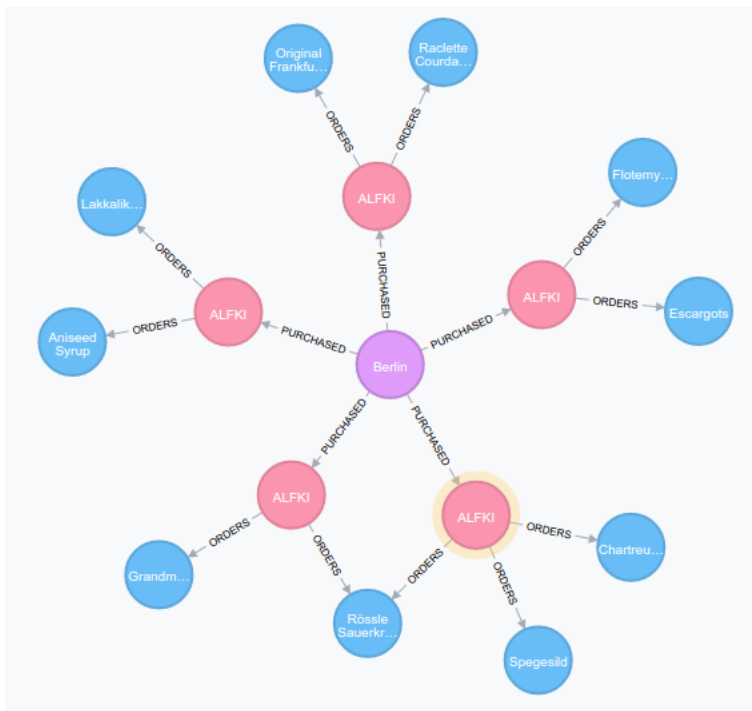
Here, the purple circle is the customer, the red circles are orders, and the blue circles are the products.

### 2.3 Exercise 6.3

Using the following query, we are able to find all orders for the customer with customerID 'ALFKI', including only the orders with at least 2 products.

```
MATCH (customer {customerID: 'ALFKI'})--(order)
MATCH p=(order)--(X)
WITH order as order, count(p) as paths
WHERE paths > 2
RETURN order
```

This returns the following graph from the database:



### 2.4 Exercise 6.4