Think Python

-Korean ver.5-

㈜ 비트시스 [대표 김동칠]

동의대학교 블록체인 인력양성 사업단

부산광역시
BUSAN METROPOLITAN CITY

부산인재평생교육진흥원
BIT Busan Institute for Talent & Lifelong Education

BitSys
행복한 꿈을 공유하다

제 13 장

웹서비스 사용하기

프로그램을 사용하여 HTTP상에서 문서를 가져와서 파싱하는 것이 익숙해지면, 다른 프로그램(즉, 브라우져에서 HTML로 보여지지 않는 것)에서 활용되도록 특별히 설계된 문서를 생성하는 것은 그다지 오래 걸리지 않는다.

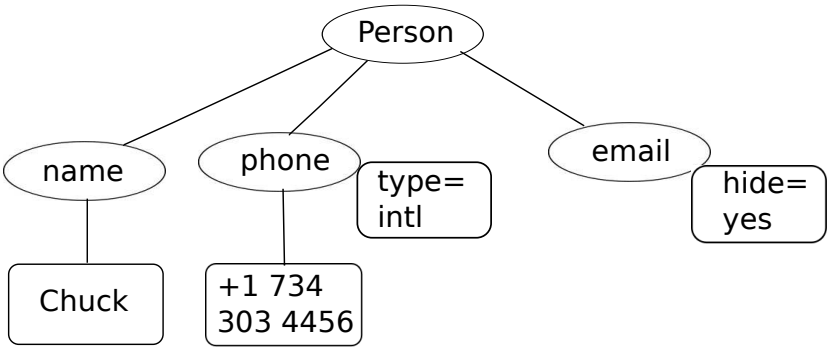
웹상에서 데이터를 교환할 때 두 가지 형식이 많이 사용된다. XML("eXtensible Markup Language")은 오랜 기간 사용되어져 왔고 문서-형식(document-style) 데이터를 교환하는데 가장 적합하다. 딕셔너리, 리스트 혹은 다른 내부 정보를 프로그램으로 서로 교환할 때, JSON(JavaScript Object Notation, `www.json.org`)을 사용한다. 두 가지 형식에 대해 모두 살펴볼 것이다.

제 1 절 XML(eXtensible Markup Language)

XML은 HTML과 매우 유사하지만, XML이 좀더 HTML보다 구조화되었다. 여기 XML 문서 샘플이 있다.

```
<person>
  <name>Chuck</name>
  <phone type="intl">
     +1 734 303 4456
   </phone>
   <email hide="yes"/>
</person>
```

종종 XML문서를 나무 구조(tree structure)로 생각하는 것이 도움이 된다. 최상단 `person` 태그가 있고, `phone` 같은 다른 태그는 부모 노드의 *자식(children)* 노드로 표현된다.



제 2 절 XML 파싱

다음은 XML을 파싱하고 XML에서 데이터 요소를 추출하는 간단한 응용프로그램이다.

```
import xml.etree.ElementTree as ET

data = '''
<person>
  <name>Chuck</name>
  <phone type="intl">
     +1 734 303 4456
   </phone>
   <email hide="yes"/>
</person>'''

tree = ET.fromstring(data)
print 'Name:',tree.find('name').text
print 'Attr:',tree.find('email').get('hide')
```

`fromstring`**을 호출하여 XML 문자열 표현을 XML 노드 '나무(tree)'로 변환한다. XML이 나무구조로 되었을 때, XML에서 데이터 일부분을 추출하기 위해서 호출하는 메쏘드가 연달아 있다.**

`find` **함수는 XML 나무를 훑어서 특정한 태그와 매칭되는 노드(node)를 검색한다. 각 노드는 텍스트, 속성(즉, hide 같은), 그리고 "자식(child)" 노드로 구성된다. 각 노드는 노드 나무의 최상단이 될 수 있다.**

```
Name: Chuck
Attr: yes
```

`ElementTree`**같은 XML 파서를 사용하는 것은 장점이 있다. 상기 예제의 XML은 매우 간단하지만, 적합한 XML에 관해서 규칙이 많이 있고, XML 구문 규칙에 얽매이지 않고** `ElementTree`**를 사용해서 XML에서 데이터를 추출할 수 있다.**

제 3 절 노드 반복하기

종종 XML이 다중 노드를 가지고 있어서 모든 노드를 처리하는 루프를 작성할 필요가 있다. 다음 프로그램에서 모든 `user` **노드를 루프로 반복한다.**

```
import xml.etree.ElementTree as ET

input = '''
<stuff>
    <users>
        <user x="2">
            <id>001</id>
            <name>Chuck</name>
        </user>
        <user x="7">
            <id>009</id>
            <name>Brent</name>
        </user>
    </users>
</stuff>'''

stuff = ET.fromstring(input)
lst = stuff.findall('users/user')
print 'User count:', len(lst)

for item in lst:
    print 'Name', item.find('name').text
    print 'Id', item.find('id').text
    print 'Attribute', item.get('x')
```

findall 메쏘드는 파이썬 리스트의 하위 나무를 가져온다. 리스트는 XML 나무에서 user 구조를 표현한다. 그리고 나서, for 루프를 작성해서 각 user 노드 값을 확인하고 name, id 텍스트 요소와 user 노드에서 x 속성도 출력한다.

```
User count: 2
Name Chuck
Id 001
Attribute 2
Name Brent
Id 009
Attribute 7
```

제 4 절 JSON(JavaScript Object Notation)

JSON 형식은 자바스크립트 언어에서 사용되는 객체와 배열 형식에서 영감을 얻었다. 하지만 파이썬이 자바스크립트 이전에 개발되어서 딕셔너리와 리스트의 파이썬 구문이 JSON 구문에 영향을 주었다. 그래서 JSON 포맷이 거의 파이썬 리스트와 딕셔너리의 조합과 일치한다.

상기 간단한 XML에 대략 상응하는 JSON으로 작성한 것이 다음에 있다.

```
{
  "name" : "Chuck",
  "phone" : {
    "type" : "intl",
    "number" : "+1 734 303 4456"
   },
```

```
   "email" : {
     "hide" : "yes"
   }
}
```

몇가지 차이점에 주목하세요. 첫째로 XML에서는 "phone" 태그에 "intl"같은 속성을 추가할 수 있다. JSON 에서는 단지 키-값 페어(key-value pair)다. 또한 XML "person" 태그는 사라지고 외부 중괄호 세트로 대체되었다.

일반적으로 JSON 구조가 XML 보다 간단하다. 왜냐하면, JSON 이 XML보다 적은 역량을 보유하기 때문이다. 하지만 JSON 이 딕셔너리와 리스트의 조합에 *직접* 매핑된다는 장점이 있다. 그리고, 거의 모든 프로그래밍 언어가 파이썬 딕셔너리와 리스트에 상응하는 것을 갖고 있어서, JSON 이 협업하는 두 프로그램 사이에서 데이터를 교환하는 매우 자연스러운 형식이 된다.

XML 에 비해서 상대적으로 단순하기 때문에, JSON 이 응용프로그램 간 거의 모든 데이터를 교환하는데 있어 빠르게 선택되고 있다.

제 5 절 JSON 파싱하기

딕셔너리(객체)와 리스트를 중첩함으로써 JSON을 생성한다. 이번 예제에서, user 리스트를 표현하는데, 각 user가 키-값 페어(key-value pair, 즉, 딕셔너리)다. 그래서 리스트 딕셔너리가 있다.

다음 프로그램에서 내장된 json 라이브러리를 사용하여 JSON을 파싱하여 데이터를 읽어온다. 이것을 상응하는 XML 데이터, 코드와 비교해 보세요. JSON은 조금 덜 정교해서 사전에 미리 리스트를 가져오고, 리스트가 사용자이고, 각 사용자가 키-값 페어 집합임을 알고 있어야 한다. JSON은 좀더 간략(장점)하고 하지만 좀더 덜 서술적(단점)이다.

```
import json

input = '''
[
  { "id" : "001",
    "x" : "2",
    "name" : "Chuck"
  } ,
  { "id" : "009",
    "x" : "7",
    "name" : "Chuck"
  }
]'''

info = json.loads(input)
print 'User count:', len(info)

for item in info:
    print 'Name', item['name']
    print 'Id', item['id']
    print 'Attribute', item['x']
```

JSON과 XML에서 데이터를 추출하는 코드를 비교하면, json.loads()을 통해서 파이썬 리스트를 얻는다. for 루프로 파이썬 리스트를 훑고, 리스트 내부의 각 항목은 파이썬 딕셔너리로 각 사용자별 다양한 정보를 추출하기 위해서 파이썬 인덱스 연산자를 사용한다. JSON을 파싱하면, 네이티브 파이썬 객체와 구조가 생성된다. 반환된 데이터가 단순히 네이티브 파이썬 구조체이기 때문에, 파싱된 JSON을 활용하는데 JSON 라이브러리를 사용할 필요는 없다.

프로그램 출력은 정확하게 상기 XML 버젼과 동일한다.

```
User count: 2
Name Chuck
Id 001
Attribute 2
Name Brent
Id 009
Attribute 7
```

일반적으로 웹서비스에 대해서 XML에서 JSON으로 옮겨가는 산업 경향이 뚜렷하다. JSON이 프로그래밍 언어에서 이미 갖고 있는 네이티브 자료 구조와 좀더 직접적이며 간단히 매핑되기 때문에, JSON을 사용할 때 파싱하고 데이터 추출하는 코드가 더욱 간단하고 직접적이다. 하지만 XML 이 JSON 보다 좀더 자기 서술적이고 XML 이 강점을 가지는 몇몇 응용프로그램 분야가 있다. 예를 들어, 대부분의 워드 프로세서는 JSON보다는 XML을 사용하여 내부적으로 문서를 저장한다.

제 6 절 API(Application Program Interfaces, 응용 프로그램 인터페이스)

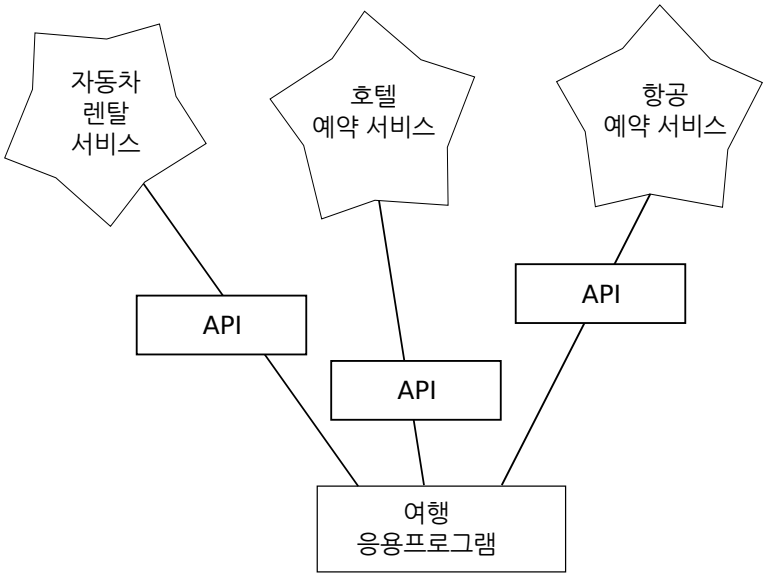
이제 HTTP를 사용하여 응용프로그램간에 데이터를 교환할 수 있게 되었다. 또한, XML 혹은 JSON을 사용하여 응용프로그램간에도 복잡한 데이터를 주고받을 수 있는 방법을 습득했다.

다음 단계는 상기 학습한 기법을 사용하여 응용프로그램 간에 "계약(contract)"을 정의하고 문서화한다. 응용프로그램-대-응용프로그램 계약에 대한 일반적 명칭은 API 응용 프로그램 인터페이스(Application Program Interface) 다. API를 사용할 때, 일반적으로 하나의 프로그램이 다른 응용 프로그램에서 사용할 수 있는 가능한 서비스 집합을 생성한다. 또한, 다른 프로그램이 서비스에 접근하여 사용할 때 지켜야하는 API (즉, "규칙")도 게시한다.

다른 프로그램에서 제공되는 서비스에 접근을 포함하여 프로그램 기능을 개발할 때, 이러한 개발법을 SOA, Service-Oriented Architecture(서비스 지향 아키텍처)라고 부른다. SOA 개발 방식은 전반적인 응용 프로그램이 다른 응용 프로그램 서비스를 사용하는 것이다. 반대로, SOA가 아닌 개발방식은 응용 프로그램이 하나의 독립된 응용 프로그램으로 구현에 필요한 모든 코드를 담고 있다.

웹을 사용할 때 SOA 사례를 많이 찾아 볼 수 있다. 웹사이트 하나를 방문해서 비행기표, 호텔, 자동차를 단일 사이트에서 예약완료한다. 호텔관련 데이터는

물론 항공사 컴퓨터에 저장되어 있지 않다. 대신에 항공사 컴퓨터는 호텔 컴퓨터와 계약을 맺어 호텔 데이터를 가져와서 사용자에게 보여준다. 항공사 사이트를 통해서 사용자가 호텔 예약을 동의할 경우, 항공사 사이트에서 호텔 시스템의 또다른 웹서비스를 통해서 실제 예약을 한다. 전체 거래(transaction)를 완료하고 카드 결재를 진행할 때, 다른 컴퓨터가 프로세스에 관여하여 처리한다.



서비스 지향 아키텍쳐는 많은 장점이 있다. (1) 항상 단 하나의 데이터만 유지관리한다. 이중으로 중복 예약을 원치 않는 호텔 같은 경우에 매우 중요하다. (2) 데이터 소유자가 데이터 사용에 대한 규칙을 정한다. 이러한 장점으로, SOA 시스템은 좋은 성능과 사용자 요구를 모두 만족하기 위해서 신중하게 설계되어야 한다.

응응프로그램이 웹상에 이용가능한 API로 서비스 집합을 만들 때, 웹서비스(web services)라고 부른다.

제 7 절 구글 지오코딩 웹서비스(Google Geocoding Web Service)

구글이 자체적으로 구축한 대용량 지리 정보 데이터베이스를 누구나 이용할 수 있게 하는 훌륭한 웹서비스가 있다. “Ann Arbor, MI” 같은 지리 검색 문자열을 지오코딩 API 에 넣으면, 검색 문자열이 의미하는 지도상에 위치와 근처 주요 지형지물 정보를 나름 최선을 다해서 예측 제공한다.

지오코딩 서비스는 무료지만 사용량이 제한되어 있어서, 상업적 응용프로그램에 API를 무제한 사용할 수는 없다. 하지만, 최종 사용자가 자유형식 입력 박스에 위치정보를 입력하는 설문 데이터가 있다면, 구글 API를 사용하여 데이터를 깔끔하게 정리하는데는 유용하다.

구글 지오코딩 API 같은 무료 API를 사용할 때, 자원 사용에 대한 지침을 준수해야 한다. 너무나 많은 사람이 서비스를 남용하게 되면, 구글은 무료 서비스를 중단하거나, 상당부분 줄일 수 있다. (When you are using a free API like Google’s

geocoding API, you need to be respectful in your use of these resources. If too many people abuse the service, Google might drop or significantly curtail its free service.

서비스에 대해서 자세한 사항을 온라인 문서를 정독할 수 있지만, 무척 간단해서 브라우저에 다음 URL을 입력해서 테스트까지 할 수 있다.

http://maps.googleapis.com/maps/api/geocode/json?sensor=false&address=Ann+Arbor%2C+MI

웹프라우저에 붙여넣기 전에, URL만 뽑아냈고 URL에서 모든 공백을 제거했는지 확인하세요.

다음은 간단한 응용 프로그램이다. 사용자가 검색 문자열을 입력하고 구글 지오코딩 API를 호출하여 반환된 JSON에서 정보를 추출한다.

```
import urllib
import json

serviceurl = 'http://maps.googleapis.com/maps/api/geocode/json?'

while True:
    address = raw_input('Enter location: ')
    if len(address) < 1 : break

    url = serviceurl + urllib.urlencode({'sensor':'false',
          'address': address})
    print 'Retrieving', url
    uh = urllib.urlopen(url)
    data = uh.read()
    print 'Retrieved',len(data),'characters'

    try: js = json.loads(str(data))
    except: js = None
    if 'status' not in js or js['status'] != 'OK':
        print '==== Failure To Retrieve ===='
        print data
        continue

    print json.dumps(js, indent=4)

    lat = js["results"][0]["geometry"]["location"]["lat"]
    lng = js["results"][0]["geometry"]["location"]["lng"]
    print 'lat',lat,'lng',lng
    location = js['results'][0]['formatted_address']
    print location
```

프로그램이 사용자로부터 검색 문자열을 받는다. 적절히 인코딩된 매개 변수로 검색문자열을 변환하여 URL을 만든다. 그리고 나서 urllib을 사용하여 구글 지오코딩 API에서 텍스트를 가져온다. 고정된 웹페이지와 달리, 반환되는 데이터는 전송한 매개변수와 구글 서버에 저장된 지리정보 데이터에 따라 달라진다.

JSON 데이터를 가져오면, json 라이브러리로 파싱하고 전송받은 데이터가 올바른지 확인하는 몇가지 절차를 거친 후에 찾고자 하는 정보를 추출한다.

프로그램 출력결과는 다음과 같다. (몇몇 JSON 출력은 의도적으로 삭제했다.)

```
$ python geojson.py
Enter location: Ann Arbor, MI
Retrieving http://maps.googleapis.com/maps/api/
  geocode/json?sensor=false&address=Ann+Arbor%2C+MI
Retrieved 1669 characters
{
    "status": "OK",
    "results": [
        {
            "geometry": {
                "location_type": "APPROXIMATE",
                "location": {
                    "lat": 42.2808256,
                    "lng": -83.7430378
                }
            },
            "address_components": [
                {
                    "long_name": "Ann Arbor",
                    "types": [
                        "locality",
                        "political"
                    ],
                    "short_name": "Ann Arbor"
                }
            ],
            "formatted_address": "Ann Arbor, MI, USA",
            "types": [
                "locality",
                "political"
            ]
        }
    ]
}
lat 42.2808256 lng -83.7430378
Ann Arbor, MI, USA
Enter location:
```

다양한 구글 지오코딩 API의 XML과 JSON을 좀더 살펴보기 위해서 **www.py4inf.com/code/geojson.py**, **www.py4inf.com/code/geoxml.py**을 다운로드 받아보기 바란다.

제 8 절 보안과 API 사용

상용업체 API를 사용하기 위해서는 일종의 "API키(API key)"가 일반적으로 필요하다. 서비스 제공자 입장에서 누가 서비스를 사용하고 있으며 각 사용자가 얼마나 사용하고 있는를 알고자 한다. 상용 API 제공업체는 서비스에 대한 무료 사용자와 유료 사용자에 대한 구분을 두고 있다. 특정 기간 동안 한 개인 사용자가 사용할 수 있는 요청수에 대해 제한을 두는 정책을 두고 있다.

때때로 API키를 얻게 되면, API를 호출할 때 POST 데이터의 일부로 포함하거나 URL의 매개변수로 키를 포함시킨다.

또 다른 경우에는 업체가 서비스 요청에 대한 보증을 강화해서 공유키와 비밀번호를 암호화된 메시지 형식으로 보내도록 요구한다. 인터넷을 통해서 서비스 요청을 암호화하는 일반적인 기술을 OAuth라고 한다. `http://www.oauth.net` 사이트에서 OAuth 프로토콜에 대해 더 많은 정보를 만날 수 있다.

트위터 API가 점차적으로 가치있게 됨에 따라 트위터가 공개된 API에서 API를 매번 호출할 때마다 OAuth 인증을 거치도록 API를 바꿔었다. 다행스럽게도 편리한 OAuth 라이브러리가 많이 있다.

그래서 명세서를 읽고 아무것도 없는 상태에서 OAuth 구현하는 것을 필할 수 있게 되었다. 이용 가능한 라이브러리는 복잡성도 다양한만틈 기능적으로도 다양하다. OAuth 웹사이트에서 다양한 OAuth 라이브러리 정보를 확인할 수 있다.

다음 샘플 프로그램으로 `www.py4inf.com/code` 사이트에서 twurl.py, hidden.py, oauth.py, twitter1.py 파일을 다운로드 받아서 컴퓨터 한 폴더에 저장한다.

프로그램을 사용하기 위해서 트위터 계정이 필요하고, 파이썬 코드를 응용프로그램으로 인증하고, 키, 암호, 토큰, 토큰 암호를 설정해야 한다. hidden.py 파일을 편집하여 4개 문자열을 파일에 적절한 변수에 저장한다.

```
def auth() :
    return { "consumer_key" : "h7L...GNg",
        "consumer_secret" : "dNK...7Q",
        "token_key" : "101...GI",
        "token_secret" : "H0yM...Bo" }
```

트위터 웹서비스는 다음 URL을 사용하여 접근한다.

`https://api.twitter.com/1.1/statuses/user_timeline.json`

하지만, 모든 비밀 정보가 추가되면, URL은 다음과 같이 보인다.

```
https://api.twitter.com/1.1/statuses/user_timeline.json?count=2
&oauth_version=1.0&oauth_token=101...SGI&screen_name=drchuck
&oauth_nonce=09239679&oauth_timestamp=1380395644
&oauth_signature=rLK...BoD&oauth_consumer_key=h7Lu...GNg
&oauth_signature_method=HMAC-SHA1
```

OAuth 보안 요구사항을 충족하기 위해 추가된 다양한 매개 변수 의미를 좀더 자세히 알고자 한다면, OAuth 명세서를 읽어보기 바란다.

트위터로 실행한 프로그램에서 oauth.py, twurl.py 두개 파일에 모든 복잡함을 감추었다. hidden.py에 암호를 설정해서 URL을 twurl.augment() 함수에 전송했고, 라이브러리 코드는 URL에 필요한 매개 변수를 추가했다.

twitter1.py 프로그램은 특정 트위터 사용자 타임라인을 가져와서 JSON 형식 문자열로 반환한다. 단순하게 문자열의 첫 250 문자만 출력한다.

```
import urllib
import twurl

TWITTER_URL='https://api.twitter.com/1.1/statuses/user_timeline.json'

while True:
    print ''
    acct = raw_input('Enter Twitter Account:')
    if ( len(acct) < 1 ) : break
    url = twurl.augment(TWITTER_URL,
        {'screen_name': acct, 'count': '2'} )
    print 'Retrieving', url
    connection = urllib.urlopen(url)
    data = connection.read()
    print data[:250]
    headers = connection.info().dict
    # print headers
    print 'Remaining', headers['x-rate-limit-remaining']
```

프로그램을 실행하면 다음 결과물을 출력한다.

```
Enter Twitter Account:drchuck
Retrieving https://api.twitter.com/1.1/ ...
[{"created_at":"Sat Sep 28 17:30:25 +0000 2013","
id":384007200990982144,"id_str":"384007200990982144",
"text":"RT @fixpert: See how the Dutch handle traffic
intersections: http:\/\/t.co\/tIiVWtEhj4\n#brilliant",
"source":"web","truncated":false,"in_rep
Remaining 178

Enter Twitter Account:fixpert
Retrieving https://api.twitter.com/1.1/ ...
[{"created_at":"Sat Sep 28 18:03:56 +0000 2013",
"id":384015634108919808,"id_str":"384015634108919808",
"text":"3 months after my freak bocce ball accident,
my wedding ring fits again! :)\n\nhttps:\/\/t.co\/2XmHPx7kgX",
"source":"web","truncated":false,
Remaining 177

Enter Twitter Account:
```

반환된 타임라인 데이터와 함께 트위터는 또한 HTTP 응답 헤더에 요청사항에 대한 메타 데이터도 반환한다. 특히 헤더에 있는 x-rate-limit-remaining정보는 한동안 서비스를 이용 못하게 되기 전까지 얼마나 많은 요청을 할 수 있는가하는 정보를 담고 있다. API 요청을 매번 할 때마다 남은 숫자가 줄어드는 것을 확인할 수 있다.

다음 예제에서, 사용자의 트위터 친구 정보를 가져와서 JSON 파싱을 하고 친구에 대한 정보를 추출한다. 파싱 후에 JSON을 가져 와서, 좀더 많은 필드를 추출할 때 데이터를 자세히 살펴보는데 도움이 되도록 문자 4개로 들여쓰기한 "보기좋은 출력(pretty-print)"을 한다.

```
import urllib
import twurl
```

```
import json

TWITTER_URL = 'https://api.twitter.com/1.1/friends/list.json'

while True:
    print ''
    acct = raw_input('Enter Twitter Account:')
    if ( len(acct) < 1 ) : break
    url = twurl.augment(TWITTER_URL,
        {'screen_name': acct, 'count': '5'} )
    print 'Retrieving', url
    connection = urllib.urlopen(url)
    data = connection.read()
    headers = connection.info().dict
    print 'Remaining', headers['x-rate-limit-remaining']
    js = json.loads(data)
    print json.dumps(js, indent=4)

    for u in js['users'] :
        print u['screen_name']
        s = u['status']['text']
        print '  ',s[:50]
```

JSON은 중첩된 파이썬 리스트와 딕셔너리 집합이기 때문에, 인덱스 연산과 `for` 루프를 조합해서 매우 적은 양의 코드로 반환된 데이터 구조를 훑어볼 수 있다.

프로그램 결과는 다음과 같다. (페이지에 맞도록 몇몇 데이터 항목을 줄였다.)

```
Enter Twitter Account:drchuck
Retrieving https://api.twitter.com/1.1/friends ...
Remaining 14
{
    "next_cursor": 1444171224491980205,
    "users": [
        {
            "id": 662433,
            "followers_count": 28725,
            "status": {
                "text": "@jazzychad I just bought one .__.",
                "created_at": "Fri Sep 20 08:36:34 +0000 2013",
                "retweeted": false,
            },
            "location": "San Francisco, California",
            "screen_name": "leahculver",
            "name": "Leah Culver",
        },
        {
            "id": 40426722,
            "followers_count": 2635,
            "status": {
                "text": "RT @WSJ: Big employers like Google ...",
                "created_at": "Sat Sep 28 19:36:37 +0000 2013",
            },
            "location": "Victoria Canada",
```

```
            "screen_name": "_valeriei",
            "name": "Valerie Irvine",
    ],
    "next_cursor_str": "1444171224491980205"
}
leahculver
   @jazzychad I just bought one .__.
_valeriei
   RT @WSJ: Big employers like Google, AT&amp;amp;T are h
ericbollens
   RT @lukew: sneak peek: my LONG take on the good &a
halherzog
   Learning Objects is 10. We had a cake with the LO,
scweeker
   @DeviceLabDC love it! Now where so I get that "etc

Enter Twitter Account:
```

출력 마지막은 drchuck 트위터 계정에서 가장 최근 친구 5명을 `for` 루프로 읽고 친구의 가장 마지막 상태 정보를 출력한다. 반환된 JSON에는 이용가능한 더 많은 데이터가 있다. 또한, 프로그램 출력을 보게 되면, 특정 계정의 "find the friends"가 일정 기간동안 실행 가능한 타임라인 질의 숫자와 다른 사용량에 제한을 두고 있음을 볼 수 있다.

이와 같은 보안 API키는 누가 트위터 API를 사용하고 어느 정도 수준으로 트위터를 사용하는지에 대해서 트위트가 확고한 신뢰를 갖게 한다. 사용량에 한계를 두고 서비스를 제공하는 방식은 단순히 개인적인 목적으로 데이터 검색을 할 수는 있지만, 하루에 수백만 API 호출로 데이터를 추출하여 제품을 개발 못하게 제한하는 기능도 동시에 한다.

제 9 절 용어정의

API: 응용 프로그램 인터페이스(Application Program Interface) - 두 응용 프로그램 컴포넌트 간에 상호작용하는 패턴을 정의하는 응용 프로그램 간의 계약.

ElementTree: XML데이터를 파싱하는데 사용되는 파이썬 내장 라이브러리.

JSON: JavaScript Object Notation- 자바스크립트 객체(JavaScript Objects) 구문을 기반으로 구조화된 데이터 마크업(markup)을 허용하는 형식.

REST: REpresentational State Transfer - HTTP 프로토콜을 사용하여 응용 프로그램 내부에 자원에 접근을 제공하는 일종의 웹서비스 스타일.

SOA: 서비스 지향 아키텍처(Service Oriented Architecture) - 응용 프로그램이 네트워크에 연결된 컴포넌트로 구성될 때.

XML: 확장 마크업 언어(eXtensible Markup Language) - 구조화된 데이터의 마크업을 허용하는 형식.

제 10 절 Exercises

Exercise 13.1 데이터를 가져와서 두 문자 국가 코드를 출력하도록 `www.py4inf.com/code/geojson.py` 혹은 `www.py4inf.com/code/geoxml.py`을 수정하세요. 오류 검사 기능을 추가하여 국가 코드가 없더라도 프로그램이 역추적(traceback)이 생성하지 않도록 하세요. 프로그램이 정상 작동하면, "Atlantic Ocean"을 검색하고 어느 국가에도 속하지 않는 지역을 처리할 수 있는지 확인하세요.

제 14 장

데이터베이스와 SQL(Structured Query Language) 사용하기

제 1 절 데이터베이스가 뭔가요?

데이터베이스(database)는 데이터를 저장하기 위한 목적으로 조직된 파일이다. 대부분의 데이터베이스는 키(key)와 값(value)를 매핑한다는 의미에서 딕셔너리처럼 조직되었다. 가장 큰 차이점은 데이터베이스는 디스크(혹은 다른 영구 저장소)에 위치하고 있어서, 프로그램 종료 후에도 정보가 계속 저장된다. 데이터베이스가 영구 저장소에 저장되어서, 컴퓨터 주기억장치(memory) 크기에 제한받는 딕셔너리보다 훨씬 더 많은 정보를 저장할 수 있다.

딕셔너리처럼, 데이터베이스 소프트웨어는 엄청난 양의 데이터 조차도 매우 빠르게 삽입하고 접근하도록 설계되었다. 컴퓨터가 특정 항목으로 빠르게 찾아갈 수 있도록 데이터베이스에 인덱스(indexes)를 추가한다. 데이터베이스 소프트웨어는 인덱스를 구축하여 성능을 보장한다.

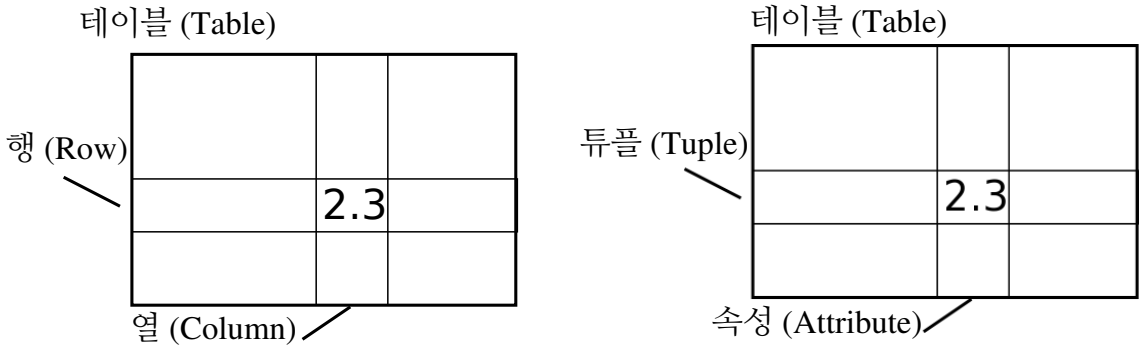
다양한 목적에 맞춰 서로 다른 많은 데이터베이스 시스템이 개발되어 사용되고 있다. Oracle, MySQL, Microsoft SQL Server, PostgreSQL, SQLite이 여기에 포함된다. 이 책에서는 SQLite를 집중해서 살펴볼 것이다. 왜냐하면 매우 일반적인 데이터베이스이며 파이썬에 이미 내장되어 있기 때문이다. 응용프로그램 내부에서 데이터베이스 기능을 제공하도록 SQLite가 다른 응용프로그램 내부에 *내장(embedded)*되도록 설계되었다. 예를 들어, 다른 많은 소프트웨어 제품이 그렇듯이, 파이어폭스 브라우져도 SQLite를 사용한다.

`http://sqlite.org/`

이번 장에서 기술하는 트위터 스파이더링 응용프로그램처럼 정보과학(Informatics)에서 마주치는 몇몇 데이터 조작 문제에 SQLite가 적합하다.

제 2 절 데이터베이스 개념

처음 데이터베이스를 볼때 드는 생각은 마치 엑셀같은 다중 시트를 지닌 스프레드쉬트(spreadsheet)같다는 것이다. 데이터베이스에서 주요 데이터 구조물은 테이블(tables), 행(rows), and 열(columns)이 된다.



관계형 데이터베이스의 기술적인 면을 설명하면 테이블, 행, 열의 개념은 관계(relation), 튜플(tuple), and 속성(attribute) 각각 형식적으로 매칭된다. 이번 장에서는 조금 덜 형식 용어를 사용한다.

제 3 절 파이어폭스 애드온 SQLite 매니저

SQLite 데이터베이스 파일에 있는 데이터를 다루기 위해서 이번장에서 주로 파이썬 사용에 집중을 하지만, 다음 웹사이트에서 무료로 이용 가능한 SQLite 데이터베이스 매니저(SQLite Database Manager)로 불리는 파이어폭스 애드온(add-on)을 사용해서 좀더 쉽게 많은 작업을 수행할 수 있다.

`https://addons.mozilla.org/en-us/firefox/addon/sqlite-manager/`

브라우져를 사용해서 쉽게 테이블을 생성하고, 데이터를 삽입, 편집하고 데이터베이스 데이터에 대해 간단한 SQL 질의를 실행할 수 있다.

이러한 점에서 데이터베이스 매니저는 텍스트 파일을 작업할 때 사용하는 텍스트 편집기와 유사하다. 텍스트 파일에 하나 혹은 몇개 작업만 수행하고자 하면, 텍스트 편집기에서 파일을 열어 필요한 수정작업을 하고 닫으면 된다. 텍스트 파일에 작업할 사항이 많은 경우는 종종 간단한 파이썬 프로그램을 작성하여 수행한다. 데이터베이스로 작업할 때도 동일한 패턴이 발견된다. 간단한 작업은 데이터베이스 매니저를 통해서 수행하고, 좀더 복잡한 작업은 파이썬으로 수행하는 것이 더 편리하다.

제 4 절 데이터베이스 테이블 생성하기

데이터베이스는 파이썬 리스트 혹은 딕셔너리보다 좀더 명확히 정의된 구조를 요구한다.[1].

[1]실질적으로 SQLite는 열에 저장되는 데이터 형식에 대해서 좀더 많은 유연성을 부여하지만, 이번 장에서는 데이터 형식을 엄격하게 적용해서 MySQL 같은 다른 관계형 데이터베이스

데이터베이스에 테이블(table)을 생성할 때, 열(column)의 명칭과 각 열(column)에 저장하는 테이터 형식을 사전에 정의해야 한다. 데이터베이스 소프트웨어가 각 열의 데이터 형식을 인식하게 되면, 데이터 형식에 따라 데이터를 저장하고 찾아오는 방법을 가장 효율적인 방식을 선택할 수 있다.

다음 url에서 SQLite에서 지원되는 다양한 데이터 형식을 살펴볼 수 있다.

http://www.sqlite.org/datatypes.html

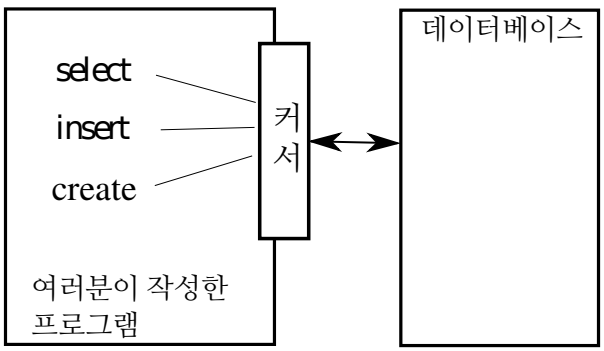
처음에는 데이터 구조를 사전에 정의하는 것이 불편하게 보이지만, 대용량의 데이터가 데이터베이스에 포함되더라도 데이터의 빠른 접근을 보장하는 잇점이 있다.

데이터베이스 파일과 데이터베이스에 두개의 열을 가진 `Tracks` 이름의 테이블을 생성하는 코드는 다음과 같다.

```
import sqlite3

conn = sqlite3.connect('music.sqlite3')
cur = conn.cursor()

cur.execute('DROP TABLE IF EXISTS Tracks ')
cur.execute('CREATE TABLE Tracks (title TEXT, plays INTEGER)')

conn.close()
```

연결 `(connect)` 연산은 현재 디렉토리 `music.sqlite3` 파일에 저장된 데이터베이스에 "연결(connection)"한다. 파일이 존재하지 않으면, 자동 생성된다. "연결(connection)"이라고 부르는 이유는 때때로 데이터베이스가 응용프로그램이 실행되는 서버로부터 분리된 "데이터베이스 서버(database server)"에 저장되기 때문이다. 지금 간단한 예제 파일의 경우에 데이터베이스가 로컬 파일 형태로 파이썬 코드 마찬가지로 동일한 디렉토리에 있다.

파일을 다루는 파일 핸들(file handle)처럼 데이터베이스에 저장된 파일에 연산을 수행하기 위해서 커서(cursor)를 사용한다. `cursor()`를 호출하는 것은 개념적으로 텍스트 파일을 다룰 때 `open()`을 호출하는 것과 개념적으로 매우 유사하다.



커서가 생성되면, `execute()` 메쏘드를 사용하여 데이터베이스 콘텐츠에 명령어 실행을 할 수 있다.

시스템에도 동일한 개념이 적용되게 한다.

데이터베이스 명령어는 특별한 언어로 표현된다. 단일 데이터베이스 언어를 학습하도록 서로 다른 많은 데이터베이스 업체 사이에서 표준화되었다.

데이터베이스 언어를 SQL(Structured Query Language 구조적 질의 언어)로 부른다.

http://en.wikipedia.org/wiki/SQL

상기 예제에서, 데이터베이스에 두개의 SQL 명령어를 실행했다. 관습적으로 데이터베이스 키워드는 대문자로 표기한다. 테이블명이나 열의 명칭처럼 사용자가 추가한 명령어 부분은 소문자로 표기한다.

첫 SQL 명령어는 만약 존재한다면 데이터베이스에서 `Tracks` 테이블을 삭제한다. 동일한 프로그램을 실행해서 오류 없이 반복적으로 `Tracks` 테이블을 생성하도록하는 패턴이다. `DROP TABLE` 명령어는 데이터베이스 테이블 및 테이블 콘텐츠 전부를 삭제하니 주의한다. (즉, "실행취소(undo)"가 없다.)

```
cur.execute('DROP TABLE IF EXISTS Tracks ')
```

두번째 명령어는 `title` 문자형 열과 `plays` 정수형 열을 가진 `Tracks`으로 명명된 테이블을 생성한다.

```
cur.execute('CREATE TABLE Tracks (title TEXT, plays INTEGER)')
```

이제 `Tracks`으로 명명된 테이블을 생성했으니, SQL `INSERT` 연산을 통해 테이블에 데이터를 넣을 수 있다. 다시 한번, 데이터베이스에 연결하여 커서(`cursor`)를 얻어 작업을 시작한다. 그리고 나서 커서를 사용해서 SQL 명령어를 수행한다.

SQL `INSERT` 명령어는 어느 테이블을 사용할지 특정한다. 그리고 나서 `(title, plays)` 포함할 필드 목록과 테이블 새로운 행에 저장될 `VALUES` 나열해서 신규 행을 정의를 마친다. 실제 값이 `execute()` 호출의 두번째 매개변수로 튜플 `( 'My Way', 15 )` 로 넘겨는 것을 표기하기 위해서 값을 물음표 `(?, ?)`로 명기한다.

```
import sqlite3

conn = sqlite3.connect('music.sqlite3')
cur = conn.cursor()

cur.execute('INSERT INTO Tracks (title, plays) VALUES ( ?, ? )',
    ( 'Thunderstruck', 20 ) )
cur.execute('INSERT INTO Tracks (title, plays) VALUES ( ?, ? )',
    ( 'My Way', 15 ) )
conn.commit()

print 'Tracks:'
cur.execute('SELECT title, plays FROM Tracks')
for row in cur :
   print row

cur.execute('DELETE FROM Tracks WHERE plays < 100')
conn.commit()
```

```
cur.close()
```

먼저 테이블에 두개 열을 삽입(INSERT)하고 commit() 명령어를 사용하여 데이터가 데이터베이스에 저장되도록 했다.

Tracks

title	plays
Thunderstruck	20
My Way	15

그리고 나서, SELECT 명령어를 사용하여 테이블에 방금 전에 삽입한 행을 불러왔다. SELECT 명령어에서 데이터를 어느 열(title, plays)에서, 어느 테이블Tracks에서 을 가져올지 명세한다. SELECT 명령문을 수행한 후에, 커서는 for문 반복을 수행하는 것과 같다. 효율성을 위해서, 커서가 SELECT 명령문을 수행할 때 데이터베이스에서 모든 데이터를 읽지 않는다. 대신에 for문에서 행을 반복해서 가져하듯이 요청시에만 데이터를 읽어온다.

프로그램 실행결과는 다음과 같다.

```
Tracks:
(u'Thunderstruck', 20)
(u'My Way', 15)
```

for 루프가 행을 두개를 읽어왔다. 각각의 행은 title로 첫번째 값을,plays로 두번째 값을 갖는 파이썬 튜플이다. title 문자열이 u'로 시작한다고 걱정하지 마라. 해당 문자열은 라틴 문자가 아닌 다국어를 저장할 수 있는 유니코드(Unicode) 문자열을 나타내는 것이다.

프로그램 마지막에 SQL 명령어를 실행 사용해서 방금전에 생성한 행을 모두 삭제(DELETE) 했기 때문에 프로그램을 반복해서 실행할 수 있다. 삭제(DELETE) 명령어는 WHERE 문을 사용하여 선택 조건을 표현할 수 있다. 따라서 명령문에 조건을 충족하는 행에만 명령문을 적용한다. 이번 예제에서 기준이 모든 행에 적용되어 테이블에 아무 것도 없게 된다. 따라서 프로그램을 반복적으로 실행할 수 있다. 삭제(DELETE)를 실행한 후에 commit()을 호출하여 데이터베이스에서 데이터를 완전히 제거했다.

제 5 절 SQL(Structured Query Language) 요약

지금까지, 파이썬 예제를 통해서 SQL(Structured Query Language)을 사용했고, SQL 명령어에 대한 기본을 다루었다. 이번 장에서는 SQL 언어를 보고 SQL 구문 개요를 살펴본다.

대단히 많은 데이터베이스 업체가 존재하기 때문에 호환성의 문제로 SQL(Structured Query Language)이 표준화되었다. 그래서, 여러 업체가 개발한 데이터베이스 시스템 사이에 호환하는 방식으로 커뮤니케이션 가능하다.

관계형 데이터베이스는 테이블, 행과 열로 구성된다. 열(column)은 일반적으로 텍스트, 숫자, 혹은 날짜 자료형을 갖는다. 테이블을 생성할 때, 열의 명칭과 자료형을 지정한다.

```
CREATE TABLE Tracks (title TEXT, plays INTEGER)
```

테이블에 행을 삽입하기 위해서 SQL INSERT 명령어를 사용한다.

```
INSERT INTO Tracks (title, plays) VALUES ('My Way', 15)
```

INSERT 문장은 테이블 이름을 명기한다. 그리고 나서 새로운 행에 놓고자 하는 열/필드 리스트를 명시한다. 그리고 나서 키워드 VALUES와 각 필드 별로 해당하는 값을 넣는다.

SQL SELECT 명령어는 데이터베이스에서 행과 열을 가져오기 위해 사용된다. SELECT 명령문은 가져오고자 하는 행과 WHERE절을 사용하여 어느 행을 가져올지 지정한다. 선택 사항으로 ORDER BY 절을 이용하여 반환되는 행을 정렬할 수도 있다.

```
SELECT * FROM Tracks WHERE title = 'My Way'
```

* 을 사용하여 WHERE 절에 매칭되는 각 행의 모든 열을 데이터베이스에서 가져온다.

주목할 점은 파이썬과 달리 SQL WHERE 절은 등식을 시험하기 위해서 두개의 등치 기호 대신에 단일 등치 기호를 사용한다. WHERE에서 인정되는 다른 논리 연산자는 <, >, <=, >=, != 이고, 논리 표현식을 생성하는데 AND, OR, 괄호를 사용한다.

다음과 같이 반환되는 행이 필드값 중 하나에 따라 정렬할 수도 있다.

```
SELECT title,plays FROM Tracks ORDER BY title
```

행을 제거하기 위해서, SQL DELETE 문장에 WHERE 절이 필요하다. WHERE 절이 어느 행을 삭제할지 결정한다.

```
DELETE FROM Tracks WHERE title = 'My Way'
```

다음과 같이 SQL UPDATE 문장을 사용해서 테이블에 하나 이상의 행 내에 있는 하나 이상의 열을 갱신(UPDATE) 할 수 있다.

```
UPDATE Tracks SET plays = 16 WHERE title = 'My Way'
```

UPDATE 문장은 먼저 테이블을 명시한다. 그리고 나서, SET 키워드 다음에 변경할 필드 리스트 와 값을 명시한다. 그리고 선택사항으로 갱신될 행을 WHERE절에 지정한다. 단일 UPDATE 문장은 WHERE절에서 매칭되는 모든 행을 갱신한다. 혹은 만약 WHERE절이 지정되지 않으면,테이블 모든 행에 대해서 갱신(UPDATE)을 한다.

네가지 기본 SQL 명령문(INSERT, SELECT, UPDATE, DELETE)은 데이터를 생성하고 유지 관리하는데 필요한 기본적인 4가지 작업을 가능케 한다.

제 6 절 데이터베이스를 사용한 트위터 스파이더링(Spidering)

이번장에서 트위커 계정을 조사하고 데이터베이스를 생성하는 간단한 스파이더링 프로그램을 작성합니다. *주의: 프로그램을 실행할 때 매우 주의하세요. 여러분의 트위터 계정 접속이 차단될 정도로 너무 많은 데이터를 가져오거나 장시간 프로그램을 실행하지 마세요.*

임의 스파이더링 프로그램이 가지는 문제점 중의 하나는 종종 중단되거나 여러번 재시작할 필요가 생긴다는 것이다. 이로 사유로 지금까지 가져온 데이터를 잃을수도 있다는 것이다. 데이터 가져오기온 처음 시점에서 항상 다시 시작하고 싶지는 않다. 그래서 데이터를 가져오면 저장하길 원한다. 프로그램이 자동으로 백업작업을 수행해서 중단된 곳에서부터 다시 가져오는 작업을 했으면 한다.

출발은 한 사람의 트위터 친구와 상태 정보를 가져오는 것에서 시작한다. 그리고, 친구 리스트를 반복하고, 향후에 가져올 수 있도록 친구 각각을 데이터베이스에 추가한다. 한 사람의 트위터 친구를 처리한 후에, 테이터베이스를 확인하고 친구의 친구 한명을 가져온다. 이것을 반복적으로 수행하고, "방문하지 않는(unvisited)" 친구를 선택하고,친구의 리스트를 가져온다. 그리고 향후 방문을 위해서 현재 리스트에서 보지 않은 친구를 추가한다.

"인기도(popularity)"를 측정하도록 데이터베이스에 특정 친구를 얼마나 자주 봤는지를 기록한다.

알고 있는 계정 리스트를 저장함으로써, 혹은 계정을 가져왔는 혹은 그렇지 않은지, 그리고 계정이 컴퓨터 하드디스크 데이터베이스에서 얼마나 인기있는지에 따라 원하는 만큼 프로그램을 멈추거나 다시 시작할 수 있다.

프로그램이 약간 복잡하다. 트위터 API를 사용한 책의 앞선 예제에서 가져온 코드에 기반하여 작성되었다.

다음이 트위터 스파이더링 응용프로그램 소스코드다.

```
import urllib
import twurl
import json
import sqlite3

TWITTER_URL = 'https://api.twitter.com/1.1/friends/list.json'

conn = sqlite3.connect('spider.sqlite3')
cur = conn.cursor()

cur.execute('''
CREATE TABLE IF NOT EXISTS Twitter
(name TEXT, retrieved INTEGER, friends INTEGER)''')

while True:
    acct = raw_input('Enter a Twitter account, or quit: ')
    if ( acct == 'quit' ) : break
    if ( len(acct) < 1 ) :
```

```
        cur.execute('SELECT name FROM Twitter WHERE retrieved = 0 LIMIT 1')
        try:
            acct = cur.fetchone()[0]
        except:
            print 'No unretrieved Twitter accounts found'
            continue

    url = twurl.augment(TWITTER_URL,
                {'screen_name': acct, 'count': '20'} )
    print 'Retrieving', url
    connection = urllib.urlopen(url)
    data = connection.read()
    headers = connection.info().dict
    # print 'Remaining', headers['x-rate-limit-remaining']
    js = json.loads(data)
    # print json.dumps(js, indent=4)

    cur.execute('UPDATE Twitter SET retrieved=1 WHERE name = ?', (acct, ) )

    countnew = 0
    countold = 0
    for u in js['users'] :
        friend = u['screen_name']
        print friend
        cur.execute('SELECT friends FROM Twitter WHERE name = ? LIMIT 1',
            (friend, ) )
        try:
            count = cur.fetchone()[0]
            cur.execute('UPDATE Twitter SET friends = ? WHERE name = ?',
                (count+1, friend) )
            countold = countold + 1
        except:
            cur.execute('''INSERT INTO Twitter (name, retrieved, friends)
                VALUES ( ?, 0, 1 )''', ( friend, ) )
            countnew = countnew + 1
    print 'New accounts=',countnew,' revisited=',countold
    conn.commit()

cur.close()
```

데이터베이스는 `spider.sqlite3` 파일에 저장되어 있다. 테이블 이름은 `Twitter`다. `Twitter` 테이블은 계정 이름에 대한 열, 계정 친구 정보를 가져왔는지 여부를 나타내는 열, 그리고 계정이 얼마나 많이 "친구추가(friended)" 되었는가 나타내는 열로 구성되었다.

프로그램의 메인 루프에서, 사용자가 트위터 계정 이름을 입력하거나 프로그램에서 나가기 위해 "끝내기(quit)"를 입력한다. 사용자가 트위터 계정을 입력하면, 친구 리스트와 상태정보도 가져온다. 만약 데이터베이스에 존재하지 않다면 데이터베이스에 친구로 추가한다. 만약 친구가 이미 리스트에 존재한다면, 데이터베이스 행으로 `friends` 필드에 추가한다.

만약 사용자가 엔터키를 누르면, 아직 가져오지 않은 다음 트위터 계정에 대해서 데이터베이스 정보를 살펴본다. 그 계정의 친구와 상태 정보를 가져오고, 데이터베이스에 추가하거나 갱신하고, `friends count`를 증가한다.

친구 리스트와 상태정보를 가져왔으면, 반환된 JSON 형식 user 항목을 반복돌려 각 사용자의 screen_name을 가져온다. 그리고 나서 SELECT 문을 사용하여 데이터베이스에 screen_name이 저장되었는지, 레코드가 존재하면 친구 숫자(friends)를 확인한다.

```
countnew = 0
countold = 0
for u in js['users'] :
    friend = u['screen_name']
    print friend
    cur.execute('SELECT friends FROM Twitter WHERE name = ? LIMIT 1',
        (friend, ) )
    try:
        count = cur.fetchone()[0]
        cur.execute('UPDATE Twitter SET friends = ? WHERE name = ?',
            (count+1, friend) )
        countold = countold + 1
    except:
        cur.execute('''INSERT INTO Twitter (name, retrieved, friends)
            VALUES ( ?, 0, 1 )''', ( friend, ) )
        countnew = countnew + 1
print 'New accounts=',countnew,' revisited=',countold
conn.commit()
```

커서가 SELECT문을 수행하면 행을 가져온다. for문으로 동일한 작업을 할 수 있지만, 단지 하나의 행(LIMIT 1)만을 가져오기 때문에, SELECT 처리 결과에서 첫번째만 가져오는 fetchone() 메쏘드를 사용한다. fetchone()은 설사 하나의 필드만 있더라도 행을 튜플(tuple)로 반환하기 때문에, [0]을 사용해서 튜플로부터 첫번째 값을 얻어 count 변수에 현재 친구 숫자를 구한다.

정상적으로 데이터를 가져오면, SQL WHERE절을 가진 UPDATE문을 사용하여 친구의 계정에 매칭되는 행에 대해서 friends 열에 추가한다. SQL에 두개의 플레이스홀더(placeholder, 물음표)가 있고, execute()의 두 매개변수가 물음표 자리에 SQL 안으로 치환될 값을 가진 두-요소 튜플이 된다.

만약 try 블록에서 코드가 작동하지 않는다면, 아마도 SELECT 문의 WHERE name = ? 절에서 매칭되는 레코드가 없기 때문이다. 그래서, except 블록에서, SQL INSERT문을 사용하여 screen_name을 가져온 적이 없고 친구 숫자를 0 으로 설정해서 친구의 screen_name을 테이블에 추가한다.

처음 프로그램을 실행하고 트위터 계정을 입력하면, 프로그램이 다음과 같이 실행된다.

```
Enter a Twitter account, or quit: drchuck
Retrieving http://api.twitter.com/1.1/friends ...
New accounts= 20  revisited= 0
Enter a Twitter account, or quit: quit
```

프로그램을 처음으로 실행하여서, 데이터베이스는 비여 있다. spider.sqlite3 파일에 데이터베이스를 생성하고, Twitter 이름의 테이블을 추가한다. 그리고 나서 친구 몇명을 가져온다. 데이터베이스가 비여있기 때문에 모든 친구를 추가한다.

이 지점에서 spider.sqlite3 파일에 무엇이 있는지를 살펴보기 위해서 간단한 데이터베이스 덤퍼(dumper)를 작성한다.

```
import sqlite3

conn = sqlite3.connect('spider.sqlite3')
cur = conn.cursor()
cur.execute('SELECT * FROM Twitter')
count = 0
for row in cur :
   print row
   count = count + 1
print count, 'rows.'
cur.close()
```

상기 프로그램은 데이터베이스를 열고 Twitter 테이블의 모든 행과 열을 선택하고 루프를 모든 행에 대해 돌려 행별로 출력한다.

앞서 작성한 트위터 스파이더를 실행한 후에 이 프로그램을 실행하면, 출력 결과는 다음과 같다.

```
(u'opencontent', 0, 1)
(u'lhawthorn', 0, 1)
(u'steve_coppin', 0, 1)
(u'davidkocher', 0, 1)
(u'hrheingold', 0, 1)
...
20 rows.
```

각 screen_name에 대해 한 행만 있다. screen_name 데이터를 가져오지 않아서 데이터베이스에 있는 모든 사람은 친구가 한명 뿐이다.

이제 데이터베이스가 트위터 계정 (drchuck)에서 친구 정보를 가져온 것을 확인했다. 프로그램을 반복적으로 실행해서, 다음과 같이 트위터 계정을 입력하는 대신에 엔터키를 눌러 "처리되지 않은" 다음 계정 친구정보를 가져오게 한다.

```
Enter a Twitter account, or quit:
Retrieving http://api.twitter.com/1.1/friends ...
New accounts= 18  revisited= 2
Enter a Twitter account, or quit:
Retrieving http://api.twitter.com/1.1/friends ...
New accounts= 17  revisited= 3
Enter a Twitter account, or quit: quit
```

엔터키를 누를 때(즉, 트위터 계정을 명시하지 않았을 때), 다음 코드가 수행된다.

```
    if ( len(acct) < 1 ) :
        cur.execute('SELECT name FROM Twitter WHERE retrieved = 0 LIMIT 1')
        try:
            acct = cur.fetchone()[0]
        except:
            print 'No unretrieved twitter accounts found'
            continue
```

SQL SELECT문을 사용해서 첫 사용자(LIMIT 1) 이름을 가져온다. "사용자를 가져왔는가"의 값은 여전히 0으로 설정되어 있다. try/except 블록 내부에 fetchone()[0] 패턴을 사용하여 가져온 데이터에서 screen_name을 추출하던가 혹은 오류 메시지를 출력하고 다시 돌아간다.

처리되지 않은 screen_name을 성공적으로 가져오면, 다음과 같이 데이터를 가져온다.

```
url = twurl.augment(TWITTER_URL, {'screen_name': acct, 'count': '20'} )
print 'Retrieving', url
connection = urllib.urlopen(url)
data = connection.read()
js = json.loads(data)

cur.execute('UPDATE Twitter SET retrieved=1 WHERE name = ?', (acct, ) )
```

데이터를 성공적으로 가져오면 UPDATE문을 사용하여 이 계정의 친구 가져오기를 완료했는지 표기하기 위해서 retrieved 열에 1 을 표시한다. 이렇게 함으로써 반복적으로 동일한 데이터를 가져오지 않게 하고 트위터 친구 네트워크를 타고 앞으로 나갈 수 있게 한다.

친구 프로그램을 실행하고 다음 방문하지 않은 친구의 친구 정보를 가져오기 위해서 두번 엔터를 누르고, 결과값을 확인하는 프로그램을 실행하면, 다음 출력값을 얻게 된다.

```
(u'opencontent', 1, 1)
(u'lhawthorn', 1, 1)
(u'steve_coppin', 0, 1)
(u'davidkocher', 0, 1)
(u'hrheingold', 0, 1)
...
(u'cnxorg', 0, 2)
(u'knoop', 0, 1)
(u'kthanos', 0, 2)
(u'LectureTools', 0, 1)
...
55 rows.
```

lhawthorn과 opencontent을 방문한 이력이 잘 기록됨을 볼 수 있다. cnxorg과 kthanos 계정은 이미 두 명의 팔로워(follower)가 있다. 친구 세명(drchuck, opencontent, lhawthorn)을 가져와서, 테이블은 55 친구 행이 생겼다.

매번 프로그램을 실행하고 엔터키를 누를 때마다, 다음 방문하지 않은(예, 다음 계정은 steve_coppin) 계정을 선택해서, 친구 목록을 가져오고, 가져온 것으로 표기하고, steve_coppin 친구 각각을 데이터베이스 끝에 추가하고 데이터베이스에 이미 추가되어 있으면 친구 숫자를 갱신한다.

프로그램의 데이터가 모두 데이터베이스 디스크에 저장되어서, 스파이더링을 잠시 보류할 수 있ek. 데이터 손실 없이 원하는만큼 다시 시작할 수 있다.

제 7 절 데이터 모델링 기초

관계형 데이터베이스의 진정한 힘은 다중 테이블과 테이블 사이의 관계를 생성할 때 생긴다. 응용프로그램 데이터를 쪼개서 다중 테이블과 두 테이블 간에 관계를 설정하는 것을 데이터 모델링(data modeling)이라고 한다. 테이블 정보와 테이블 관계를 표현하는 설계 문서를 데이터 모델(data model)이라고 한다.

데이터 모델링(data modeling)은 상대적으로 고급 기술이여서 이번 장에서는 관계형 데이터 모델링의 가장 기본적인 개념만을 소개한다. 데이터 모델링에 대한 좀더 자세한 사항은 다음 링크에서 시작해 볼 수 있다.

http://en.wikipedia.org/wiki/Relational_model

트위터 스파이더 응용프로그램으로 단순히 한 사람의 친구가 몇명인지 세는 대신에, 모든 관계 리스트를 가지고서 특정 계정에 팔로잉하는 모든 사람을 찾는다.

모두 팔로잉하는 계정을 많이 가지고 있어서, 트위터(Twitter) 테이블에 단순히 하나의 열만을 추가해서는 해결할 수 없다. 그래서 친구를 짝으로 추적할 수 있는 새로운 테이블을 생성한다. 다음이 간단하게 상기 테이블을 생성하는 방식이다.

```
CREATE TABLE Pals (from_friend TEXT, to_friend TEXT)
```

drchuck을 팔로잉하는 사람을 마주칠 때마다, 다음과 같은 형식의 행을 삽입한다.

```
INSERT INTO Pals (from_friend,to_friend) VALUES ('drchuck', 'lhawthorn')
```

drchuck 트위터 피드에서 친구 20명을 처리하면서, "drchuck"을 첫 매개변수로 가지는 20개 레코드를 삽입해서 데이터베이스에 중복되는 많은 문자열이 생길 것이다.

문자열 데이터 중복은 데이터베이스 정규화(database normalization) 모범 사례(berst practice)를 위반하게 만든다. 기본적으로 데이터베이스 정규화는 데이터베이스에 결코 한번 이상 동일한 문자열을 저장하지 않는다. 만약 한번 이상 데이터가 필요하다면, 그 특정 데이터에 대한 숫자 키(key)를 생성하고, 그 키를 사용하여 실제 데이터를 참조한다.

실무에서, 문자열이 컴퓨터 주기억장치나 디스크에 저장되는 정수형 자료보다 훨씬 많은 공간을 차지하고 더 많은 처리시간이 비교나 정렬에 소요된다. 항목이 단지 수백개라면, 저장소나 처리 시간이 그다지 문제되지 않는다. 하지만, 데이터베이스에 수백만명의 사람 정보와 1억건 이상의 링크가 있다면, 가능한 빨리 데이터를 스캔하는 것이 정말 중요하다.

앞선 예제에서 사용된 Twitter 테이블 대신에 People로 명명된 테이블에 트위커 계정을 저장한다. People 테이블은 트위터 사용자에 대한 행과 관련된 숫자 키를 저장하는 추가 열(column)이 있다. SQLite는 데이터 열의 특별한 자료형(INTEGER PRIMARY KEY)을 이용하여 테이블에 삽입할 임의 행에 대해서 자동적으로 키값을 추가하는 기능이 있다.

다음과 같이 추가적인 id 열을 가진 People 테이블을 생성할 수 있다.

```
CREATE TABLE People
    (id INTEGER PRIMARY KEY, name TEXT UNIQUE, retrieved INTEGER)
```

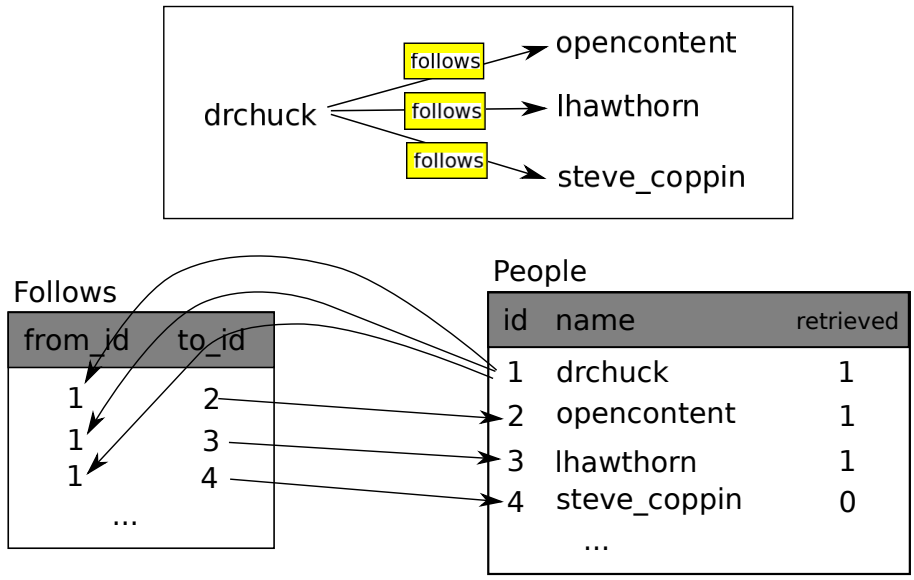
People 테이블의 각 행에서 친구 숫자를 더 이상 유지관리하고 있지 않음을 주목하세요. id 열 자료형으로 INTEGER PRIMARY KEY 선택할 때 함축되는 의미는 다음과 같다., 사용자가 삽입하는 각 행에 대해서 SQLite가 자동으로 유일한 숫자 키를 할당하고 관리하게 한다. UNIQUE 키워드를 추가해서 SQLite에 name에 동일한 값을 가진 두 행을 삽입하지 못하게 한다.

상기 Pals 테이블을 생성하는 대신에, 데이터베이스에 from_id, to_id 두 정수 자료형 열을 지닌 Follows 테이블을 생성한다. Follows 테이블은 from_id과 to_id의 *조합*으로 테이블이 유일하다는 제약사항도 가진다. (즉, 중복된 행을 삽입할 수 없다.)

```
CREATE TABLE Follows
    (from_id INTEGER, to_id INTEGER, UNIQUE(from_id, to_id) )
```

테이블에 UNIQUE절을 추가한다는 의미는 레코드를 삽입할 때 데이터베이스에서 지켜야하는 규칙 집합을 의사소통하는 것이다. 잠시 후에 보겠지만, 프로그램상에 편리하게 이러한 규칙을 생성한다. 이러한 규칙 집합은 실수를 방지하게 하고 코드를 작성을 간결하게 한다.

본질적으로 Follows 테이블을 생성할 때, "관계(relationship)"를 모델링하여 한 사람이 다른 사람을 "팔로우(follow)"하고 이것을 (a) 사람이 연결되어 있고, (b) 관계을 방향성이 나타나도록 숫자를 짝지어 표현한다.



제 8 절 다중 테이블을 가지고 프로그래밍

테이블 두개, 주키(primary key)와 앞서 설명된 참조 키를 사용하여 트위터 스파이더링 프로그램을 다시 작성한다. 다음은 새로운 버젼 프로그램 코드다.

```
import urllib
import twurl
import json
import sqlite3

TWITTER_URL = 'https://api.twitter.com/1.1/friends/list.json'

conn = sqlite3.connect('friends.sqlitesqlite3')
cur = conn.cursor()

cur.execute('''CREATE TABLE IF NOT EXISTS People
    (id INTEGER PRIMARY KEY, name TEXT UNIQUE, retrieved INTEGER)''')
cur.execute('''CREATE TABLE IF NOT EXISTS Follows
    (from_id INTEGER, to_id INTEGER, UNIQUE(from_id, to_id))''')

while True:
    acct = raw_input('Enter a Twitter account, or quit: ')
    if ( acct == 'quit' ) : break
    if ( len(acct) < 1 ) :
        cur.execute('''SELECT id, name FROM People
            WHERE retrieved = 0 LIMIT 1''')
        try:
            (id, acct) = cur.fetchone()
        except:
            print 'No unretrieved Twitter accounts found'
            continue
    else:
        cur.execute('SELECT id FROM People WHERE name = ? LIMIT 1',
            (acct, ) )
        try:
            id = cur.fetchone()[0]
        except:
            cur.execute('''INSERT OR IGNORE INTO People (name, retrieved)
                VALUES ( ?, 0)''', ( acct, ) )
            conn.commit()
            if cur.rowcount != 1 :
                print 'Error inserting account:',acct
                continue
            id = cur.lastrowid

    url = twurl.augment(TWITTER_URL,
       {'screen_name': acct, 'count': '20'} )
    print 'Retrieving account', acct
    connection = urllib.urlopen(url)
    data = connection.read()
    headers = connection.info().dict
    print 'Remaining', headers['x-rate-limit-remaining']

    js = json.loads(data)
    # print json.dumps(js, indent=4)

    cur.execute('UPDATE People SET retrieved=1 WHERE name = ?', (acct, ) )

    countnew = 0
    countold = 0
```

```
    for u in js['users'] :
        friend = u['screen_name']
        print friend
        cur.execute('SELECT id FROM People WHERE name = ? LIMIT 1',
            (friend, ) )
        try:
            friend_id = cur.fetchone()[0]
            countold = countold + 1
        except:
            cur.execute('''INSERT OR IGNORE INTO People (name, retrieved)
                VALUES ( ?, 0)''', ( friend, ) )
            conn.commit()
            if cur.rowcount != 1 :
                print 'Error inserting account:',friend
                continue
            friend_id = cur.lastrowid
            countnew = countnew + 1
        cur.execute('''INSERT OR IGNORE INTO Follows (from_id, to_id)
            VALUES (?, ?)''', (id, friend_id) )
    print 'New accounts=',countnew,' revisited=',countold
    conn.commit()

cur.close()
```

프로그램이 다소 복잡해 보인다. 하지만, 테이블을 연결하기 위해서 정수형 키를 사용하는 패턴을 보여준다. 기본적인 패턴은 다음과 같다.

1. **주키(primary key)와 제약 사항을 가진 테이블을 생성한다.**
2. **사람(즉, 계정 이름)에 대한 논리 키가 필요할 때 사람에 대한** `id` **값이 필요하다. 사람 정보가 이미** `People` **테이블에 존재하는지에 따라, (1)** `People` **테이블에 사람을 찾아서 그 사람에 대한** `id` **값을 가져오거나, (2) 사람을** `People` **테이블에 추가하고 신규로 추가된 행의** `id` **값을 가져온다.**
3. **"팔로우(follow)" 관계를 잡아내는 행을 추가한다.**

이들 각각을 순서대로 다룰 것이다.

8.1 데이터베이스 테이블의 제약사항

테이블 구조를 설계할 때, 데이터베이스 시스템에 몇 가지 규칙을 설정할 수 있다. 이러한 규칙은 실수를 방지하고 잘못된 데이터가 테이블에 들어가는 것을 막는다. 테이블을 생성할 때:

```
cur.execute('''CREATE TABLE IF NOT EXISTS People
    (id INTEGER PRIMARY KEY, name TEXT UNIQUE, retrieved INTEGER)''')
cur.execute('''CREATE TABLE IF NOT EXISTS Follows
    (from_id INTEGER, to_id INTEGER, UNIQUE(from_id, to_id))''')
```

`People` **테이블에** `name` **칼럼이 유일(**`UNIQUE`**)함을 나타낸다.** `Follows` **테이블의 각 행에서 두 숫자 조합은 유일하다는 것도 나타낸다. 하나 이상의 동일한 관계를 추가하는 것 같은 실수를 이러한 제약 사항을 통해서 방지한다.**

다음 코드에서 이런 제약사항의 장점을 확인할 수 있다.

```
cur.execute('''INSERT OR IGNORE INTO People (name, retrieved)
    VALUES ( ?, 0)''', ( friend, ) )
```

INSERT 문에 OR IGNORE 절을 추가해서 만약 특정 INSERT가 "name이 유일(unique)해야 한다"를 위반하게 되면, 데이터베이스 시스템은 INSERT를 무시한다. 데이터베이스 제약 사항을 안전망으로 사용해서 무언가가 우연히 잘못되지 않게 방지한다.

마찬가지로, 다음 코드는 정확히 동일 Follows관계를 두번 추가하지 않는다.

```
cur.execute('''INSERT OR IGNORE INTO Follows
    (from_id, to_id) VALUES (?, ?)''', (id, friend_id) )
```

다시 한번, Follows 행에 대해 지정한 유일한 제약사항을 위반하게 되면 INSERT 시도를 무시하도록 데이터베이스에 지시한다.

8.2 레코드를 가져오거나 삽입하기

사용자가 트위터 계정을 입력할 때, 만약 계정이 존재한다면, id 값을 찾아야 한다. 만약 People 테이블에 계정이 존재하지 않는다면, 레코드를 삽입하고 삽입된 행에서 id 값을 얻어와야 한다.

이것은 매우 일반적인 패턴이고, 상기 프로그램에서 두번 수행되었다. 가져온 트위터 JSON 사용자(user) 노드에서 screen_name을 추출할 때, 친구 계정의 id를 어떻게 찾는지 코드가 보여준다.

시간이 지남에 따라 점점 더 많은 계정이 데이터베이스에 존재할 것 같기 때문에, SELECT문을 사용해서 People 레코드가 존재하는지 먼저 확인한다.

try 구문 내에서 모든 것이 정상적으로 잘 작동하면[2], fetchone()을 사용하여 레코드를 가져와서, 반환된 튜플의 첫번째 요소만 읽어오고 friend_id에 저장한다.

만약 SELECT가 실패하면, fetchone()[0] 코드도 실패하고 제어권은 except 블록으로 이관된다.

```
        friend = u['screen_name']
        cur.execute('SELECT id FROM People WHERE name = ? LIMIT 1',
            (friend, ) )
        try:
            friend_id = cur.fetchone()[0]
            countold = countold + 1
        except:
            cur.execute('''INSERT OR IGNORE INTO People (name, retrieved)
                VALUES ( ?, 0)''', ( friend, ) )
            conn.commit()
            if cur.rowcount != 1 :
```

[2] 일반적으로, 문장이 "만약 모든 것이 잘 된다면"으로 시작하면, 코드는 필히 try/except를 필요로 한다.

```
            print 'Error inserting account:',friend
            continue
        friend_id = cur.lastrowid
        countnew = countnew + 1
```

except 코드에서 끝나게 되면, 행을 찾지 못해서 행을 삽입해야 한다는 의미가 된다. INSERT OR IGNORE를 사용해서 오류를 피하고 데이터베이스에 진정으로 갱신하기 위해서 commit()을 호출한다. 데이터베이스에 쓰기가 수행된 후에, 얼마나 많은 행이 영향을 받았는지 확인하기 위해서 cur.rowcount로 확인한다. 단일 행을 삽입하려고 했는데, 영향을 받은 행의 숫자가 1과 다르다면, 그것은 오류다.

삽입(INSERT)이 성공하면, cur.lastrowid를 살펴보고 신규로 생성된 행의 id 칼럼에 무슨 값이 대입되었는지 알 수 있다.

8.3 친구관계 저장하기

트위터 사용자와 JSON 친구에 대한 키값을 알게되면, 다음 코드로 두 개의 숫자를 Follows 테이블에 삽입하는 것은 간단하다.

```
cur.execute('INSERT OR IGNORE INTO Follows (from_id, to_id) VALUES (?, ?)',
    (id, friend_id) )
```

데이터베이스를 생성할 때 유일(unique)한 제약조건과 INSERT문에 OR IGNORE을 추가함으로써 데이터베이스 스스로 관계를 두번 삽입하는 것을 방지하도록 한 것에 주목한다.

다음에 프로그램의 샘플 실행 결과가 있다.

```
Enter a Twitter account, or quit:
No unretrieved Twitter accounts found
Enter a Twitter account, or quit: drchuck
Retrieving http://api.twitter.com/1.1/friends ...
New accounts= 20  revisited= 0
Enter a Twitter account, or quit:
Retrieving http://api.twitter.com/1.1/friends ...
New accounts= 17  revisited= 3
Enter a Twitter account, or quit:
Retrieving http://api.twitter.com/1.1/friends ...
New accounts= 17  revisited= 3
Enter a Twitter account, or quit: quit
```

drchuck 계정으로 시작해서, 프로그램이 자동적으로 다음 두개의 계정을 선택해서 데이터베이스에 추가한다.

다음은 프로그램 수행을 완료한 후에 People과 Follows 테이블에 첫 몇개의 행이다.

```
People:
(1, u'drchuck', 1)
(2, u'opencontent', 1)
(3, u'lhawthorn', 1)
```

```
(4, u'steve_coppin', 0)
(5, u'davidkocher', 0)
55 rows.
Follows:
(1, 2)
(1, 3)
(1, 4)
(1, 5)
(1, 6)
60 rows.
```

People 테이블의 id, name, visited 필드와 Follows 테이블 양 끝에 관계 숫자를 볼 수 있다. People 테이블에서, 사람 첫 세명을 방문해서, 데이터를 가져온 것을 볼 수 있다. Follows 테이블의 데이터는 drchuck(사용자 1)이 첫 다섯개 행에 보여진 모든 사람에 대해 친구임을 나타낸다. 이것은 당연한데 왜냐하면 처음 가져와서 저장한 데이터가 drchuck의 트위터 친구들이기 때문이다. Follows 테이블에서 좀더 많은 행을 출력하면, 사용자 2, 3 혹은 그 이상의 친구를 볼 수 있다.

제 9 절 세 종류의 키

지금까지 데이터를 다중 연결된 테이블에 넣고 키(keys)를 사용하여 행을 연결하는 방식으로 데이터 모델을 생성했는데, 키와 관련된 몇몇 용어를 살펴볼 필요가 있다. 일반적으로 데이터베이스 모델에서 세가지 종류의 키가 사용된다.

- 논리 키(logical key)는 "실제 세상"이 행을 찾기 위해서 사용하는 키다. 데이터 모델 예제에서, name 필드는 논리키다. 사용자에 대해서 screen_name이고, name 필드를 사용하여 프로그램에서 여러번 사용자 행을 찾을 수 있다. 논리 키에 UNIQUE 제약 사항을 추가하는 것이 의미있다는 것을 종종 이해하게 된다. 논리 키는 어떻게 바깥 세상에서 행을 찾는지 다루기 때문에, 테이블에 동일한 값을 가진 다중 행이 존재한다는 것은 의미가 없다.

- 주키(primary key)는 통상적으로 데이터베이스에서 자동 대입되는 숫자다. 프로그램 밖에서는 일반적으로 의미가 없고, 단지 서로 다른 테이블에서 행을 열결할 때만 사용된다. 테이블에 행을 찾을 때, 통상적으로 주키를 사용해서 행을 찾는 것이 가장 빠르게 행을 찾는 방법이다. 주키는 정수형이어서, 매우 적은 저장공간을 차지하고 매우 빨리 비교 혹은 정렬할 수 있다. 이번에 사용된 데이터 모델에서 id 필드가 주키의 한 예가 된다.

- 외부 키(foreign key)는 일반적으로 다른 테이블에 연관된 행의 주키를 가리키는 숫자다. 이번에 사용된 데이터 모델의 외부 키의 사례는 from_id다.

주키 id필드명을 호출하고, 항상 외부키에 임의 필드명에 접미사로 _id 붙이는 명명규칙을 사용한다.

제 10 절 JOIN을 사용하여 데이터 가져오기

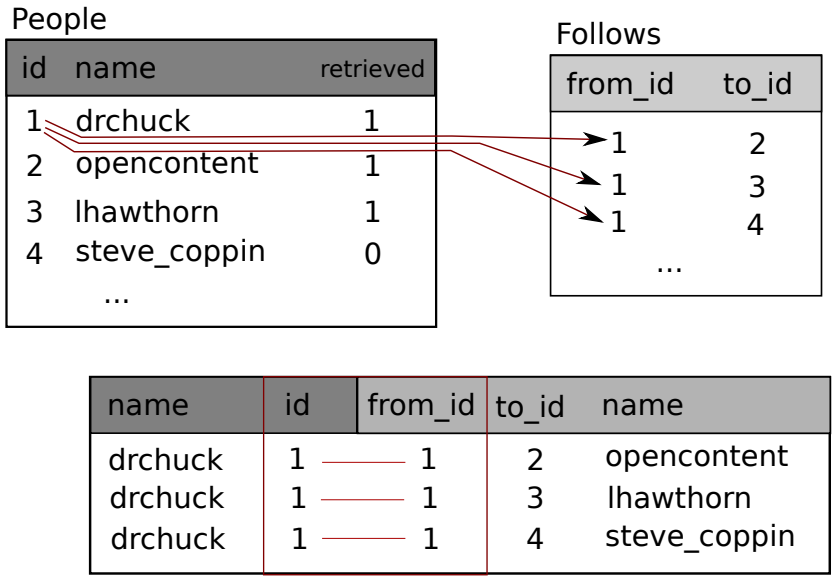
데이터 정규화 규칙을 따라서, 데이터를 주키와 외부키로 연결된 두개의 테이블로 나누어서, 테이블 데이터를 다시 합치기 위해서 SELECT를 작성할 필요가 있다.

SQL은 JOIN절을 사용해서 테이블을 다시 연결한다. JOIN절에서 테이블 사이의 행을 다시 연결할 필드를 지정한다.

다음은 JOIN절을 가진 SELECT 예제이다.

```
SELECT * FROM Follows JOIN People
    ON Follows.from_id = People.id WHERE People.id = 1
```

JOIN절은 Follows와 People 테이블에서 선택하는 필드를 나타낸다. ON절은 어떻게 두 테이블이 합쳐지는지를 나타낸다. Follows에서 행을 선택하고 People에서 행을 추가하는데, Follows 테이블의 from_id와 People 테이블의 id 값은 동일하다.



JOIN 결과는 People 테이블 필드와 Follows 테이블에서 매칭되는 필드를 가진 "메타-행(meta-row)"을 생성한다. People 테이블 id 필드와 Follows 테이블 from_id 사이에 하나 이상의 매칭이 존재한다면, JOIN은 필요하면 데이터를 중복하면서, 행에 매칭되는 짝 각각에 대해 메타-행을 생성한다.

다중 테이블 트위터 프로그램을 수차례 수행한 후에 데이터베이스에 있는 데이터를 가지고 다음 코드가 시연한다.

```
import sqlite3

conn = sqlite3.connect('spider.sqlite3')
cur = conn.cursor()

cur.execute('SELECT * FROM People')
count = 0
print 'People:'
for row in cur :
```

```
   if count < 5: print row
   count = count + 1
print count, 'rows.'

cur.execute('SELECT * FROM Follows')
count = 0
print 'Follows:'
for row in cur :
   if count < 5: print row
   count = count + 1
print count, 'rows.'

cur.execute('''SELECT * FROM Follows JOIN People
    ON Follows.from_id = People.id WHERE People.id = 2''')
count = 0
print 'Connections for id=2:'
for row in cur :
   if count < 5: print row
   count = count + 1
print count, 'rows.'

cur.close()
```

이 프로그램에서 People과 Follows 테이블을 먼저 보여주고, 함께 JOIN된 데이터 일부분을 보여준다.

다음이 프로그램 출력이다.

```
python twjoin.py
People:
(1, u'drchuck', 1)
(2, u'opencontent', 1)
(3, u'lhawthorn', 1)
(4, u'steve_coppin', 0)
(5, u'davidkocher', 0)
55 rows.
Follows:
(1, 2)
(1, 3)
(1, 4)
(1, 5)
(1, 6)
60 rows.
Connections for id=2:
(2, 1, 1, u'drchuck', 1)
(2, 28, 28, u'cnxorg', 0)
(2, 30, 30, u'kthanos', 0)
(2, 102, 102, u'SomethingGirl', 0)
(2, 103, 103, u'ja_Pac', 0)
20 rows.
```

People과 Follows 테이블에서 칼럼을 볼 수 있고, 마지막 행의 집합은 JOIN 절을 가진 SELECT문의 결과다.

마지막 SELECT문에서, "opencontent" (즉 People.id=2)를 친구로 가진 계정을 찾는다.

마지막 SELECT "메타-행"의 각각에서 첫 두 열은 Follows 테이블에서, 3번째부터 5번째 열은 People 테이블에서 가져왔다. 두번째 열(Follows.to_id)과 세번째 열(People.id)은 join된 "메타-열"에서 매칭됨을 볼 수 있다.

제 11 절 요약

이번 장은 파이썬에서 데이터베이스 사용 기본적인 개요에 대해 폭넓게 다루었다. 데이터를 저장하기 위해서 파이썬 딕셔너리나 일반적인 파일보다 데이터베이스를 사용하여 코드를 작성하는 것이 훨씬 복잡하다. 그래서, 만약 작성하는 응용프로그램이 실질적으로 데이터베이스 역량을 필요하지 않는다면 굳이 데이터베이스를 사용할 이유는 없다. 데이터베이스가 특히 유용한 상황은 (1) 큰 데이터셋에서 작은 임의적인 갱신이 많이 필요한 응용프로그램을 작성할 때 (2) 데이터가 너무 커서 딕셔너리에 담을 수 없고 반복적으로 정보를 검색할 때, (3) 한번 실행에서 다음 실행 때까지 데이터를 보관하고, 멈추고, 재시작하는데 매우 긴 실행 프로세스를 갖는 경우다.

많은 응용프로그램 요구사항을 충족시키기 위해서 단일 테이블로 간단한 데이터베이스를 구축할 수 있다. 하지만, 대부분의 문제는 몇개의 테이블과 서로 다른 테이블간에 행이 연결된 관계를 요구한다. 테이블 사이 연결을 만들 때, 좀더 사려깊은 설계와 데이터베이스의 역량을 가장 잘 사용할 수 있는 데이터베이스 정규화 규칙을 따르는 것이 중요하다. 데이터베이스를 사용하는 주요 동기는 처리할 데이터의 양이 많기 때문에, 데이터를 효과적으로 모델링해서 프로그램이 가능하면 빠르게 실행되게 만드는 것이 중요하다.

제 12 절 디버깅

SQLite 데이터베이스에 연결하는 파이썬 프로그램을 개발할 때 하나의 일반적인 패턴은 파이썬 프로그램을 실행하고 SQLite 데이터베이스 브라우저를 통해서 결과를 확인하는 것이다. 브라우저를 통해서 빠르게 프로그램이 정상적으로 작동하는지를 확인할 수 있다.

SQLite에서 두 프로그램이 동시에 동일한 데이터를 변경하지 못하기 때문에 주의가 필요하다. 예를 들어, 브라우저에서 데이터베이스를 열고 데이터베이스에 변경을 하고 "저장(save)"버튼을 누르지 않는다면, 브라우져는 데이터베이스 파일에 "락(lock)"을 걸구, 다른 프로그램이 파일에 접근하는 것을 막는다. 특히, 파일이 잠겨져 있으면 작성하고 있는 파이썬 프로그램이 파일에 접근할 수 없다.

해결책은 데이터베이스가 잠겨져 있어서 파이썬 코드가 작동하지 않는 문제를 피하도록 파이썬에서 데이터베이스에 접근하려 시도하기 전에 데이터베이스 브라우져를 닫거나 혹은 File 메뉴를 사용해서 브라우져 데이터베이스를 닫는 것이다.

제 13 절 용어정의

속성(attribute): 튜플 내부에 값의 하나. 좀더 일반적으로 "열", "칼럼", "필드"로 불린다.

제약(constraint): 데이터베이스가 테이블의 필드나 행에 규칙을 강제하는 것. 일반적인 제약은 특정 필드에 중복된 값이 없도록 하는 것(즉, 모든 값이 유일해야 한다.)

커서(cursor): 커서를 사용해서 데이터베이스에서 SQL 명령어를 수행하고 데이터베이스에서 데이터를 가져온다. 커서는 네트워크 연결을 위한 소켓이나 파일의 파일 핸들러와 유사하다.

데이터베이스 브라우져(database browser): 프로그램을 작성하지 않고 직접적으로 데이터베이스에 연결하거나 데이터베이스를 조작할 수 있는 소프트웨어.

외부 키(foreign key): 다른 테이블에 있는 행의 주키를 가리키는 숫자 키. 외부키는 다른 테이블에 저장된 행사이에 관계를 설정한다.

인텍스(index): 테이블에 행이 추가될 때 정보 검색하는 것을 빠르게 하기 위해서 데이터베이스 소프트웨어가 유지관리하는 추가 데이터.

논리 키(logical key): "외부 세계"가 특정 행의 정보를 찾기 위해서 사용하는 키. 사용자 계정 테이블의 예로, 사람의 전자우편 주소는 사용자 데이터에 대한 논리 키의 좋은 후보자가 될 수 있다.

정규화(normalization): 어떠한 데이터도 중복이 없도록 데이터 모델을 설계하는 것. 데이터베이스 한 장소에 데이터 각 항목 정보를 저장하고 외부키를 이용하여 다른 곳에서 참조한다.

주키(primary key): 다른 테이블에서 테이블의 한 행을 참조하기 위해서 각 행에 대입되는 숫자 키. 종종 데이터베이스는 행이 삽입될 때 주키를 자동 삽입하도록 설정되었다.

관계(relation): 튜플과 속성을 담고 있는 데이터베이스 내부 영역. 좀더 일반적으로 "테이블(table)"이라고 한다.

튜플(tuple): 데이터베이스 테이블에 단일 항목으로 속성 집합이다. 좀더 일반적으로 "행(row)"이라고 한다.