Using Database with Python\_notes

#####Lecture 1: Object oriented programming#####

Methods: functions that are part of the object

dictionaries can be appended to lists [ {} {} {} {} ]

keys = [‘Title’, ‘Director’, ‘Rating’, ‘Running time’]

for item in movies:

print ‘---------‘

for key in keys:

print key, ‘:’, item[key]

print ‘-------‘

#Object: self-contained Code and Data

Object approach: divide and conquer

String objects, integer object, dictionary, list

#Objects hide detail, allow us to ignore detail of the “rest of the program”

#Terminology

Class: template, method or message: eg. bark(), field or attribute: a bit of data in a class, object or instance: a particular thing in the class.

Class PartyAnimal #class is a reserved word

X = 0

def party(self) #First variable is usually “self”#

self.x = self.x + 1 #self = X#

print “So far”, self. X

an = PartyAnimal() #Create a PartyAnimal object # # return the object, and put back in “an”#

an.party() #Tell the object to run the party() code#

an.party() # = PartyAnimal. Party(an)#

an.party() # run party() \*within\* the object an

#dir() #This command can list out capabilities

#Object life cycle

Constructor

def\_init\_(self):

print “I am constructed”

Destructor

def\_del\_(self):

print “I am destructed”, self.x

#Inheritance

Subclasses

Eg.

class FootballFan(PartyAnimal): #inherit everything PartyAnimal has, aka extension#

points = 0

def touchdown(self):

self.points = self.points +7

self.party()

print self.name, “points”, self.points

##Lecture 2##

install database browser

relational database

#relation (table): contains tuples + attributes

tuple (or row): set of fields represents an object

attribute (column or field): elements of data corresponding to the object

#Data Schema #Data model, talk with database applications (API)

SQL: structured query language (API between an application and a database system)

* Create a table
* Retrieve some data
* Insert data
* Delete data

##Take dictionary and put in database

Import sqlite3

Cur = conn.cursor() #cursor is a connection object

Cur.execute(‘’’

CREATE TABLE Counts (email TEXT, count INTEGER)’’’)

Count = cur.fetchone()[0] #get one row from the list

Sqlstr = ‘SELECT email, count FROM Counts ORDER BY count DESC #descending# LIMIT 10’ #Only give the 1st 10 values

##Lecture 3: Data models and relational SQL

Link tables together

Don’t put the same string data in twice, use a relationship instead

Deal with replicated data in columns, allow very long strings

Data modeling exercise

Table

primary key (refer to a number, one key to every row, end points of an arrow we are pointing to eg. ID)

logical key (to look up this row from outside world, usually used in a where clause eg. TITLE)

foreign key (starting point of the arrow)

logical key

SQLite:

PK (primary key),

AI(auto increment)

#Insert data to the table

Insert into Artist (name) values (‘Led Zepplin’) #ID field is an auto-generative field

Insert into Genre(name) values (‘Rock’);

Insert into Album (title, artist\_id) values (‘Who Made Who’, 2)

Insert into Track (title, rating, len, count,album\_id, genre\_id) values (‘Black Dog’, 5, 297, 0, 2, 1)

#Constructing data with JOIN

Relational Power (build a “web”, tables linked by “foreign keys”)

JOIN operation (links across several tables, tell the JOIN how to use keys with an ON clause)

select Album.title, Artist.name from Album join Artist on Album.artist\_id = Artist.id

select Album.title, Album.artist\_id, Artist.id, Artist.name from Album join Artist on Album.artist\_id = Artist.id

All combinations (erasing the purple part): Joining two tables without an ON clause gives all possible combinations of rows

##Multi-table Tracks Demo

Connect xml and databases

<plist> #property lists

<dict> # dictionaries, key value pairs# #dictionaries in dictionaries# #Each track is represented by a dictionary#

import xml.etree.ElementTree as ET

import sqlite3

conn = sqlite3.connect (‘trackdb.sqlite’) #conn is the connection to sqlite

cur = conn.cursor() # cursor is the way we actually send comments

#Make some tables

Cur.execute(‘’’

CREATE TABLE IF NOT EXISTS Artist ( ##This does nothing if the table already exists

Id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,

Nam TEXT UNIQUE ##Logical key##

) ‘’’)

CREATE TABLE IF NOT EXISTS Album

…

…

…

Fname = raw\_input (‘Enter file name: ‘)

If (len(fname) < 1 ): fname = ‘Library.xml’

#Create a function to go though the dictionary, parse key value from the dictionary object

Def lookup(d, key):

Found = False

For child in d:

If found : return child.text

If child.tag == ‘key’ and child.text == key:

Found = True

Return None

Stuff = ET. Parse (fname) #Use the ET function

All = stuff.findall(‘dict/dict/dict’) #look for three layer dicts, 3rd level#

print ‘Dict count:’, len(all)

for entry in all: ## loop through all dinctionaries xml bits##

if (lookup (entry, ‘Track ID’ is None) : continue

name = lookup (entry, ‘Name’)

artist = lookup (entry, ‘Artist’)

if name is None or artist is None or album is None:

continue

print name, artist, album, count, rating, length

cur.execute (‘’’INSERT OR INGORE INTO Artist (name)

VALUES (?)’’’, (artist, )) ##? Here correspond to the “artist” in the tuple##

cur.execute (‘SELECT id FROM Artist WHERE name = ?’, (artist, ))

artist\_id = cur.fetchone()[0] ##use the cursor to grab one row, the first thing (ID guy)## ##The primary key#

cur.execute(‘’’INSERT OR IGNORE INTO Album (title, artist\_id)

VALUES (?, ?)’’’, (album, artist\_id))

Cur.execute(‘SELECT id FROM Album WHERE title = ? ‘, (album, ))

Album\_id = cur.fetchone()[0]

Cur.execute(‘’’INSERT OR REPLACE INTO Track

(title, album\_id, len, rating, count)

VALUES (?,?,?,?,?)’’’,

(name, album\_id, length, rating, count))

Conn.commit() #write the whole thing to disk

####Lecture 4####

#Many-to-Many Relationships is SQL#

Eg. Books and authors, need a junction table, courses and users

Junction table: membership table

Member: 2 foreign keys – (user\_id, course\_id)

Member: primary key (user\_id, course\_id) combination will be unique

SELECT User.name, Member.role, Course.title FROM User JOIN Member JOIN Course ON Member.user\_id = User.id AND Member.course\_id = Course.id ORDER BY Course.title, Member.role DESC, User.name

Many-to-Many rooster demo

Junction table, decompose to a pair to many table

Additional column: eg. role

conn = sqlite3.connect (‘rosterdb.sqlite’) # make a connection to sql

cur.executescript (‘’’ # use multiple SQL strands c.f. execute(), executescript()

DROP TABLE IF EXISTS User;

DROP TABLE IF EXISTS Member;

CREATE TABLE Member (

User\_id INTERGER,

Course\_id INTERGER;

PRIMARY KEY (user\_id, course\_id) #junction table: two primary keys

)

‘’’)

Fname = raw\_input(“Enter file name:”)

If (len(fname)<1) : fname = ‘roster\_data.json’

str\_data = open(fname).read()

json\_data = json.loads(str\_data)

for entry in json\_data:

name = entry[0];

title = entry [1];

cur.execute(‘’’INSERT OR IGNORE INTO User (name)

VALUES (?)’’’, (name, )) ##take advantage of Unique constraint, only insert for the 1st time seen##

Cur.execute(‘SELECT id FROM User WHERE name = ?’, (name, ))

User\_id = cur.fetchone()[0]

cur.execute(‘’’INSERT OR REPLACE INTO Member

(user\_id, course\_id) VALUES (?, ?)’’’,

(user\_id, course\_id))

Junction table: duplication allowed, two primary keys, combination of two keys is unique, sometimes put another element in the table extra data, two foreign keys, no primary key (composite key)

Auto increment, primary keys

###Lecture 5####

# Databases and visualization#

Gencoding

Multi-step data analysis

Gather data first, have raw data in the database - > clean/process - > analysis

Page rank algorithms

Web crawler, index building, searching

Mailing list – data crawling

Crawl the archive of a mailing list

Gmane.org

Eg google.geo coding API

Geoload.py

Import Urllib, sqlite3, json, time

Conn = sqlite3.connect(‘geodata.sqlite’)

Cur = conn.cursor()

(Buffer(address), ) 2nd parameter is a tuple,

js = json.loads(str(data))

conn.commit()

time.sleep(1)