CMPUT 291 - File and Database Management (Fall 2019)

Introduction to Python bsddb3

bsddb3 is a Python module that acts as a wrapper for the Oracle/Sleepcat C API. It will provide all the functions required to complete Mini-Project 2, but has some pitfalls that we will outline later. Note: You will not be able to test any of this on the lab machines until the scheduled updates on March 20th/2013.

If you are installing bsddb3 on your home machine, you can do one of the following:

- Debian or Debian-like operating systems can grab the package: python3-bsddb3 with sudo apt-get install python3-bsddb3.
- Download the source and Oracle Berkeley. You may also need gcc installed. To compile the source of bsddb3, run: python Setup.py install.

1. Establish a connection

Establishing a connection to our database is easy!

```
from bsddb3 import db
#Get an instance of BerkeleyDB
database = db.DB()
database.open("mydatabase")
database.close()
```

By default, your database will open in read only. If the database cannot be opened, it will throw exception type: bsddb3.db.DBInvalidArgError

So what's going on here? First we are constructing a database object. Next we can invoke any db method on that object, such as opening the database. This database can be either a Btree, Hash, Queue, or Recno. More on what methods are available can be found here.

2. Iterating over the cursor object

Once you have opened your database, iterating over the cursor is rather straight forward.

```
database.open("mydatabase")
cur = database.cursor()
iter = cur.first()
while iter:
 print(iter)
 iter = cur.next()
cur.close()
database.close()
```

So what's going on here? We are grabbing the first entry in the database and iterating over it by calling the next row with cur.next(). More on what the cursor object can do can be found here.

3. The pitfall to watch out for

If you ran the last example in Python 3 over a database, you might of noticed output like so:

```
(b'cooler', b'20')
(b'apple', b'5')
(b'diamond', b'8000')
```

Looks like a normal tuple, no? Well, yes, but what is that 'b' in front of the values? They are byte literals. This due to compatibility between older and newer versions of Python. Why does this matter? Let's look at the following example:

```
cur = database.cursor()
#row = (b'cooler', b'20')
row = cur.first()
# column = b'cooler'
column = row[0]
if column == "cooler":
 print("Matches!")
else:
 print("Nope!")
```

This will print Nope! So it is important you consider what formats you are dealing with. How can we fix the above code? We can decode it.

```
cur = database.cursor()
#row = (b'cooler', b'20')
row = cur.first()
# column = b'cooler'
column = row[0]
if column.decode("utf-8") == "cooler":
 print("Matches!")
else:
 print("Nope!")
```

The above example should now print Matches! Remember, only values can be decoded, not the tuple itself. if you are using Python 2.7, you do not need to account for this hack.

4. Creating a database

Creating a database of varying types isn't much different from the examples above either.

```
database = db.DB()
database.open("mydatabase", None, db.DB_HASH, db.DB_CREATE)
```

So what's going on here?

- First argument is the file name.
- Second argument is the database name within the file. If you want to use multiple databases with the same physical file, database-ception.
- Third argument is the database type. In this case we are creating a hash database. You can also use db.DB_BTREE, db.DB_HEAP, db.DB_QUEUE, or db.DB_RECNO.
- Fourth argument is setting the flag to create the database.

If you want to use memory, you can set the first and second arguments to type None.

5. Inserting data

For this part, we'll go over two different, equally effective ways of inserting data. It all depends on your preference and code layout.

Inserting by cursor object:

```
cur.put(b'apple', "red",db.DB_KEYFIRST)
```

Inserting by database object:

```
database.put(b'test', "test2")
```

There is not much difference between the two, except for the extra argument on the cursor object. This is how the cursor should behave when inserting the new data. More can be read in the documentation on flags and what their purpose is. This is not to say database.put doesn't have any flags as well. You may also notice that our key values are byte literals again. If we do not do this, we will get a TypeError: Bytes or Integer object expected for key, str found.

6. Getting data by index

```
result = cur.get(b'apple', db.DB_FIRST)
```

Here we are getting the first row by index 'apple'. More flags are available.

```
result = database.get(b'test')
```

Same thing here by default. You can always read up on more flags.

For more information, check out the documentation on dbcursor and db.

If you are having any issues with the module, need clarification, or have corrections then please use the forums. I will do my best to respond to your questions in a timely manner. Good luck and happy coding in Python!

7. More information

Check out the lab slides and consult Python bsddb documentation (you may find this document useful as well).

This tutorial is initially prepared by Kyle Richelhoff for CMPUT291 and is maintained by Davood Rafiei. Send your comments and updates to Davood.

Last modified: Thursday, 31 October 2019, 9:26 PM

◄ Mini-Project 1   Jump to...   Berkeley DB Tutorial (C & Java) ►