Unit 2 / Lesson 1 / Assignment 5

# Improve the Error Handling

Estimated Time: 1-2 hours

By this point you can retrieve a snippet that exists. If the snippet doesn't exist you are probably getting an ugly traceback. Although this is an obvious indication to a user that something has gone wrong, it's hardly courteous. Let's see what we can do about that.

Your code for fetch a row probably looks something like the following:

```
row = cursor.fetchone()
connection.commit()
return row[0]
```

When you use this code to retrieve something that doesn't exist, you will get an error, something like this:

```
Traceback (most recent call last):
  File "snippets.py", line 60, in <module>
    main()
  File "snippets.py", line 56, in main
    snippet = get(**arguments)
  File "snippets.py", line 30, in get
    return row[0]
TypeError: 'NoneType' object is not subscriptable
```

This is a traceback. It is the Python interpreter's way of being incredibly helpful in the face of something it isn't sure how to deal with. If you have trouble with a Python program and want to ask for help, be sure to copy and paste the entire text from the word "Traceback" down to the error message at the end; experienced programmers on newsgroups and mailing lists (or, of course, your mentor!) will be far better able to advise you.

In this case, you can go straight to the appropriate line (it might not be line 30 in your program) and see that it's trying to examine index 0 of a 'NoneType' object. There's only one NoneType object, and it's called None; so you can correctly conclude that `row` is None, instead of being a tuple. This is what fetchone() returns if no row exists.

Just before the `return row[0]` line, add this:

```
    if not row:
        # No snippet was found with that name.
        return "404: Snippet Not Found"
```

What should be done when an error happens? Take a look back at your docstring. Are you still happy with what you decided earlier? Should you return None? Should you raise an error? Should you freeze the user's computer, phone the police, and report that an illegal action was performed? Test it out, make sure the program does what you expect it to, and commit that to source control.

## Coping with duplicates

There's a similar issue when you put a snippet: if it already exists, you get back an ugly error. Once again, this gives you a full traceback – something like this:

```
Traceback (most recent call last):
  File "snippets.py", line 60, in <module>
    main()
  File "snippets.py", line 54, in main
    put(**arguments)
  File "snippets.py", line 14, in put
    cursor.execute("insert into snippets values (%s, %s)", (name, snip
psycopg2.IntegrityError: duplicate key value violates unique constrain
DETAIL:  Key (keyword)=(asdf) already exists.
```

In this case, the error is raised right in the `cursor.execute()` call, so the way to handle this is with `try` and `except`. Replace that line with this block:

```
try:
    command = "insert into snippets values (%s, %s)"
    cursor.execute(command, (name, snippet))
except psycopg2.IntegrityError as e:
    connection.rollback()
    command = "update snippets set message=%s where keyword=%s"
    cursor.execute(command, (snippet, name))
```

The obvious solution to this situation is to overwrite instead of inserting. The `cursor.rollback` method basically acts as an undo function to get the database back to its original state. Once that's done, you can construct the `update` command and use it to overwrite the snippet.

This solution works beautifully in the simple case you're working with here, although it has unsolvably difficult problems in large systems. If you're curious, it's usually called a MERGE or UPSERT (for UPdate/inSERT) operation.

Commit this version of the code, and then try to break your own program. Is there anything else that can go wrong? What happens? Should your program cope with those issues, or is a traceback fine?

Now that you've identified and made a first attempt at fixing these errors it's time to take a look at a more excellent way of handling Postgres errors.

## Using a cursor as a context manager

So far, you've explicitly committed or rolled back anywhere you use the database. This makes it all too easy to forget to roll back, accidentally commit too much, or end the program without committing. This is likely to cause you some unpleasant debugging headaches. The psycopg2 module gives us an easier way to handle cursors. In your `get()` function, replace these lines:

```
cursor = connection.cursor()
cursor.execute("select message from snippets where keyword=%s", (n
row = cursor.fetchone()
connection.commit()
```

With these:

```
with connection, connection.cursor() as cursor:
    cursor.execute("select message from snippets where keyword=%s"
    row = cursor.fetchone()
```

Aside from being one line shorter and one indentation level deeper, what's the difference? Simply this: At the backtab after the `with` block, the transaction is guaranteed to be either committed (if everything seemed to work) or rolled back (if an exception was thrown).

The cursor object is what is known as a **context manager**. When context managers are used in a `with` block they automatically perform some cleanup actions when the block is exited.

## Try it!

- Refactor your `put` method to use context managers to commit or rollback changes to the database. This should be an internal change with no visible effect on the program's running.
- Test your code again and make sure it's still doing the right thing in all cases.
- Don't forget to commit to source control.

☆ ☆ ☆ ☆ ☆ · Report a typo or other issue

✓ Mark as completed

Previous

Next