An extension to DBAPI 2.0 for easier SQL queries

Martin Blais

EWT LLC / Madison Tyler

http://furius.ca/antiorm/

Introduction

```
connection = dbapi.connect(...)
cursor = connection.cursor()

cursor.execute("""

  SELECT * FROM Users WHERE username = 'raymond'

  """)
```

Escaping Values

```
name = 'raymond'
cursor.execute("""

  SELECT * FROM Users WHERE username = %s

  """, (name,))
```

A common mistake is to forget to call with a tuple or a dict:

```
cursor.execute("""

  SELECT * FROM Users WHERE username = %s

  """, name)  <--- this fails
```

Escaping Values

```
name = 'raymond'
cursor.execute("""

  SELECT * FROM Users WHERE username = %s

  """, (name,))
```

A common mistake is to forget to call with a tuple or a dict:

```
cursor.execute("""

  SELECT * FROM Users WHERE username = %s

  """, name)  <--- this fails
```

String Interpolation Pitfalls

Another mistake is to use string interpolation:

```
cursor.execute("""

  SELECT * FROM Users WHERE username = %s

  """ % name)
```

ERROR!

The resulting query is missing the quotes around the values:

```
  SELECT * FROM Users WHERE username = raymond
```

String Interpolation Pitfalls

And you cannot fix this by hand:

```
cursor.execute("""

  SELECT * FROM Users WHERE username = '%s'

  """ % name)
```

ERROR!

The resulting query is missing the quotes around the values:

```
SELECT * FROM Users WHERE username = 'ray's cat'
```

String Interpolation Pitfalls

Using `repr()` will not help either:

```
cursor.execute("""

  SELECT * FROM Users WHERE username = %s

  """ % repr(name))
```

ERROR!

Escaping syntax is database-specific:

```
SELECT * FROM Users WHERE username = 'ray''s cat'
```

DBAPI Must Escape Values

You absolutely *must* let DBAPI deal with the escaping of values.

The escaping syntax for

- string constants *
- timestamps
- dates
- blobs
- (other SQL data types?)

depends on the database backend.

Non-escaped Substitutions

What if you need to format non-escaped variables?

```
  SELECT email, phone FROM Users
    WHERE username = 'raymond'

cursor.execute("""

  SELECT %s, %s FROM Users WHERE username = %s

  """, (col1, col2, name))  <--- will not work
```

Non-escaped Substitutions

What if you need to format non-escaped variables?

```
  SELECT email, phone FROM Users
    WHERE username = 'raymond'
```

```
cursor.execute("""

  SELECT %s, %s FROM Users WHERE username = %%s

  """ % (col1, col2), (name,))   <-- two steps!
```

- Because of the string interpolation step, you have to use `%%s` for the escaped values
- Specifying the parameters in the right order becomes tricky

Lists and format-specifiers

Sometimes you want to render variable-length lists:

```
cursor.execute("""

  INSERT INTO Users (%s, %s)
             VALUES (%%s, %%s)

  """ % ("email", "phone"), values)
```

```
cursor.execute("""

  INSERT INTO Users (%s, %s, %s)
             VALUES (%%s, %%s, %%s)

  """ % ("email", "phone", "address"), values)
```

Lists and format-specifiers

Sometimes you want to render variable-length lists:

```
cursor.execute("""

  INSERT INTO Users (%s, %s)
             VALUES (%%s, %%s)

  """ % ("email", "phone"), values)


cursor.execute("""

  INSERT INTO Users (%s, %s, %s)
             VALUES (%%s, %%s, %%s)

  """ % ("email", "phone", "address"), values)
```

Lists and format-specifiers

```
cursor.execute("""

  INSERT INTO Users (%s)
             VALUES (%s)

  """ % (','.join(columns),
         ','.join(["%%s"] * 2)),
  values)
```

And in the real world it gets uglier. . .

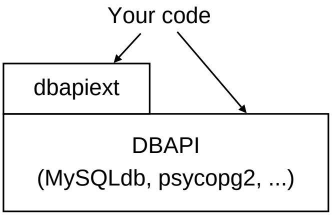
When you write real-world queries (instead of Mickey-mouse example queries), it gets even messier:

```
cursor.execute("""
  SELECT %s FROM %s
    WHERE %s > %%s
      AND %s < %%s
    LIMIT %s %s
  """ % (','.join(columns), "Users",
         "age", "age", 10, "DESC"),
  (18, 60))
```

DBAPI's `Cursor.execute()` method interface is inconvenient to use!

My Proposed Solution: `dbapiext`

- Provide a very simple extension that gets rid of the pitfalls of `execute()`
- Make it much easier to write queries
- A single pure Python module, no changes to your DBAPI
- Support a number of DBAPI implementations
- No external dependencies



New format specifier (%X)

We provide a replacement for `execute()`, and we introduce a new format specifier for escaped arguments: `%X` (capital X)

```
cursor.execute_f(
  "INSERT INTO Users (username) VALUES (%X)",
  name)
```

You can now mix vanilla and escaped values in the arguments, and you are not forced to use a tuple anymore:

```
cursor.execute_f(
  "INSERT INTO Users (%s) VALUES (%X)",
  "username", name)
```

Lists are Recognized and Understood

Lists are automatically joined with commas:

```
columns = ["username", "email", "age"]
cursor.execute_f("""

    INSERT INTO Users (%s)
               VALUES (...)

  """, columns, ...)
```

```
    INSERT INTO Users (username, email, age)
               VALUES (...)
```

Lists are Recognized and Understood

This also works for escaped arguments:

```
values = ["Warren", "w@b.com", 76]
cursor.execute_f("""

  INSERT INTO Users (%s)
            VALUES (%X)

 """, columns, values)
```

```
  INSERT INTO Users (username, email, age)
            VALUES ('Warren', 'w@b.com', 76)
```

- Values are escaped individually and then comma-joined

Dictionaries are Recognized and Understood

Dictionaries are rendered as required for `UPDATE` statements:

- Comma-separated `<name> = <value>` pairs
- Values are escaped automatically

```
UPDATE languages
  SET id = 3, brazil = 'portuguese'
```

```
values = {"id": 3,
          "brazil": "portuguese"}
cursor.execute_f("""

  UPDATE languages SET %X",

""", values)
```

(Suggestion by D. Mertz)

Keywords Arguments are Supported

```
cursor.execute_f("""

   SELECT %(table)s FROM %s
     WHERE id = %(id)X

""", column_names, table=tablename, id=42)
```

- Provide a useful way to recycle arguments
  (i.e. a table or column name that occurs multiple times)
- Positional and keyword arguments can be used simultaneously

Performance and Remarks

- The extension massages your query in a form that can be digested by DBAPI's `Cursor.execute()`
- We cache as much of the preprocessing as possible (similar to `re`, `struct`)
    - You can cache your queries at load time with `qcompile()`.
- I *lied* in my examples, you have to use it like this (if monkey-patching `Cursor` fails):

```
    execute_f(cursor, """
        ...
```

Performance and Remarks

- The extension massages your query in a form that can be digested by DBAPI's `Cursor.execute()`
- We cache as much of the preprocessing as possible (similar to `re`, `struct`)
    - You can cache your queries at load time with `qcompile()`.
- I *lied* in my examples, you have to use it like this (if monkey-patching `Cursor` fails):

```
execute_f(cursor, """
    ...
```

Final Thoughts

Ideally, we would want to automatically parse the SQL queries and determine which arguments should be quoted

- A lot more work
- Would have to be done at load time for performance reasons

Questions

`dbapiext` is part of a package named `antiorm`

antiorm homepage:
`http://furius.ca/antiorm/`

Questions?