

# An extension to DBAPI 2.0 for easier SQL queries

Martin Blais

PyCon 2007

# Introduction

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

General form:

```
cursor.execute(<string>, <tuple-or-dict>)
```

# Cursor.execute() interface

Common mistake #1:

```
cursor.execute(  
    "INSERT INTO Users (username) VALUES (%s)" %  
    var_username)
```

# Cursor.execute() interface

## Common mistake #1:

```
cursor.execute(  
    "INSERT INTO Users (username) VALUES (%s)" %  
    var_username)
```

# Cursor.execute() interface

## Common mistake #2:

```
cursor.execute(  
    "INSERT INTO Users (username) VALUES ('%s')" %  
    var_username)
```

## Common mistake #3:

```
cursor.execute(  
    "INSERT INTO Users (username) VALUES (%s)" %  
    repr(var_username))
```

# Cursor.execute() interface

## Common mistake #2:

```
cursor.execute(  
    "INSERT INTO Users (username) VALUES ('%s')" %  
    var_username)
```

## Common mistake #3:

```
cursor.execute(  
    "INSERT INTO Users (username) VALUES (%s)" %  
    repr(var_username))
```

## Cursor.execute() interface

You *must* let DBAPI do its database-specific escaping of values:

```
cursor.execute(  
    "INSERT INTO Users (username) VALUES (%s)",  
    (var_username,))
```

String constants, timestamps, dates, etc. ; Formats vary depending on the database.

Problems:

- Two lists of parameters is error-prone
- You have to provide a tuple or a dict for the argument
- It does not understand lists
- You can't leverage the power of keyword arguments

## Cursor.execute() interface

You *must* let DBAPI do its database-specific escaping of values:

```
cursor.execute(  
    "INSERT INTO Users (username) VALUES (%s)",  
    (var_username,))
```

String constants, timestamps, dates, etc. ; Formats vary depending on the database.

Problems:

- Two lists of parameters is error-prone
- You have to provide a tuple or a dict for the argument
- It does not understand lists
- You can't leverage the power of keyword arguments



# Introduction

With this work:

- Provide a simple extension that gets rid of the pitfalls
- Make it much easier to write queries
- A single pure Python module
- Support a number of DBAPI implementations
- Deals only with query writing

## Cursor.execute() interface

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 LIMIT %s
    """ % ('.', '.join(columns)', "Users",
           "age", 10),
           (18,))
```

- Because of string interpolation, you have to **double-escape** the format specifiers for the escaped values
- The parameters in the strings are in a **different order** than the function arguments (easy to make mistakes!)

## New format specifier (%S)

We provide a new `execute()` method, which supports a format specifier for escaped arguments: `%S` (capital S)

```
cursor.execute_f(  
    "INSERT INTO Users (username) VALUES (%S)",  
    var_username)
```

You can now mix vanilla and escaped values in the arguments:

```
cursor.execute_f(  
    "INSERT INTO Users (%s) VALUES (%S)",  
    "username", var_username)
```

# Lists are 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 understood

This also works for escaped arguments:

```
columns = ["username", "email", "age"]
values = [var_username, var_email, var_age]
cursor.execute_f(
    "INSERT INTO Users (%s) VALUES (%S)",
    columns, values)
```

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

- Values are escaped individually and then comma-joined

# Dictionaries are understood

Dictionaries are rendered as required for UPDATE statements:

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

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

```
values = {"id": 3,  
          "brazil": "portuguese"}
```

```
cursor.execute_f("UPDATE languages SET %S",  
                 values)
```

(Suggestion by D. Mertz)

# Positional and Keywords Arguments

Positional and keyword arguments can be used simultaneously:

```
cursor.execute_f("""  
  
    SELECT %(table)s FROM %s  
    WHERE id = %(id)s  
  
""", column_names, table=tablename, id=42)
```

- You can recycle arguments this way  
(i.e. a table or column name that occurs multiple times)

# Performance and Remarks

- The extension only massages your query in a form that can be digested by DBAPI's `Cursor.execute()`
- I lied slightly in my examples, you have to use it like this:

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

- We cache as much of the preprocessing as possible (similar to `re.struct`)
  - You can cache your queries at load time with `qcompile()`.



# Future work

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?