

A_simple ORM_SQL_for_python

This simple Object Relational Mapper is adapted from SQLAlchemy and SQLAlchemyObject. You need to create a database manually first.

```
from __future__ import generators
import e32db, re
db = e32db.Dbms()
dbv = e32db.Db_view()
db.open(u'C:\\test.db')

# Some helping classes (need more in next version)
class String:
    pass

class Integer:
    pass

class Float:
    pass

class column:
    def __init__(self, coltype):
        self.coltype = coltype

class Mapper(object):
    def __init__(self, id=None, **kw):
        if id is None:
            self.id = self._insert(**kw)
        else:
            self.id = id
    def _insert(self, **kw):
        names = ','.join(kw.keys())
        values = ','.join(self.quote(k,v) for k,v in kw.items())
        tablename = self.__class__.__name__
        q = u"INSERT INTO %s(%s) VALUES (%s)" % (tablename, names, values)
        db.execute(q)
        # get last insert ID
        dbv.prepare(db, u'SELECT id FROM '+tablename+' ORDER BY id DESC')
        dbv.first_line()
        dbv.get_line()
        return dbv.col(1)
    def __getattr__(self, name):
        if name in self.mapping.__dict__:
            q = 'SELECT '+name+' FROM '+self.__class__.__name__
            q += ' WHERE id='+str(self.id)
            dbv.prepare(db, unicode(q))
            dbv.first_line()
            dbv.get_line()
            return dbv.col(1)
        else:
            return self.__dict__[name]
    def __repr__(self):
        return '<%s id=%d>' % (self.__class__.__name__, self.id)
    def quote(self, name, value):
        if self.mapping.__dict__[name].coltype == String:
            return "'%s'" % value.replace("'", "'") # encode single quote
        else:
            return str(value)
    def __setattr__(self, name, value):
        if name in self.mapping.__dict__:
            q = 'UPDATE '+self.__class__.__name__+' SET '+name+'='
```

A_simple ORM_SQL_for_python

```
        q += self.quote(name, value) + " WHERE id=" + str(self.id)
        db.execute(unicode(q))
    else:
        self.__dict__[name] = value
def set(self, **kw):
    q = "UPDATE "+self.__class__.__name__+" SET "
    for k, v in kw.items():
        q += k+'='+self.quote(k,v)+','
    q = q[:-1]+" WHERE id=%s" % self.id
    db.execute(unicode(q))
def delete(self):
    q = 'DELETE FROM '+self.__class__.__name__+" WHERE id=" + str(self.id)
    db.execute(unicode(q))
    self.id = None
def dict(self):
    names = [k for k in self.mapping.__dict__ if not k.startswith('__')]
    q = 'SELECT '+','.join(names)+' FROM '+self.__class__.__name__
    q += ' WHERE id=' + str(self.id)
    dbv.prepare(db, unicode(q))
    dbv.first_line()
    dbv.get_line()
    dct = {'id': self.id}
    for i in range(dbv.col_count()):
        dct[names[i]] = dbv.col(i+1)
    return dct
def select(cls, where=None, orderby=None):
    q = 'SELECT id FROM '+cls.__name__
    if where:
        q += ' WHERE '+where
    if orderby:
        q += ' ORDER BY '+orderby
    dbv = e32db.Db_view() # need its own cursor
    dbv.prepare(db, unicode(q))
    dbv.first_line()
    for i in range(dbv.count_line()):
        dbv.get_line()
        yield cls(dbv.col(1))
        dbv.next_line()
select = classmethod(select)
```

Here's how you create your class.

```
class Phone(Mapper):
    class mapping:
        # doesn't need id = column(Integer)
        name = column(String)
        edition = column(Integer)
```