Assignment 3 – Molecule Database Library

Version 1.00 (last update: Mar. 7, 12:00)
Changes highlighted in yellow and green
Due date: Tue, Mar 21, 9:00 AM

Summary and Purpose

For this assignment, you will be creating a SQL Database and accompanying Python code to store molecules, elements, bonds, and atoms.

Deliverables

You will be submitting:

1) A file called **`mol.h`** that contains your typedefs, structure definitions and function prototypes (see below).
2) A file called **`mol.c`** that contains your c function code.
3) A python library call MolDisplay.py that generates SVG images of molecules.
4) A file called **`molsql.py`** that contains a number of python classes, functions, methods and variables to support the database operation of the project.
5) A **`makefile`** that contains the instructions for the following targets:
    a. **`mol.o`** – a position independent (**`-fpic`**) object code file created from **`mol.c`**.
    b. **`libmol.so`** – a shared library (-shared) created from **`mol.o`**.
    c. **`molecule_wrap.c`** and **`molecule.py`** a pair of files that provide a Python interface to your C code (these are generated using the **`swig`** program base on the instructor supplied **`molecule.i`** file.
    d. **`molecule_wrap.o`** – an object file that is an object library to interface with your C code.
    e. **`_molecule.so`** – a shared object library used by molecule.py to interface between C and Python.
    f. **`clean`** – this target should delete all **`.o`**, **`.so`** and executable files.

All compilation must be done with the **`-std=c99 -Wall -pedantic`** options and produce no warnings or errors. The **`makefile`** must correctly identify all relevant dependencies and include them as prerequisites in its rules.

You will submit all of your work via git to the School's **`gitlab`** server. (As per instructions in the labs.) Under no condition will an assignment be accepted by any other means.

All files should be managed in a single directory (don't organize your files into subdirectories based on types; sorry).

This is an individual assignment. Any evidence of code sharing will be investigated and, if appropriate, adjudicated using the University's Academic Integrity rules.

Setting up your work environment and using git:

Decide on your development environment. You can work on your own machine, the SoCS VM, no machine, or the SoCS ssh servers. However, you must test your code on the SoCS servers or SoCS VM as this is where it will be evaluated. If your code does not work properly on the SoCS systems, then it does not work.

```
git clone https://gitlab.socs.uoguelph.ca/2750W23/skremer/A3
```

But, use your own login ID instead of `skremer`.

Then, `cd A3`, to move to the assignment directory.

Work in the A3 directory. Use the command:

```
git add filename
```

For each file that you create as part of your code (see deliverables).

Every so often (especially if you get something working), use the command:

```
git commit -a
```

to commit your changes.

Use the command:

```
git tag -a Mar10a -m 'implemented the first two functions'
```

You will need to use a tag like "`Mar10a`" to request help with your code via the course help page.

And then make sure to use:

```
git push --all --follow-tags
```

to push everything to the server with the tag included.

If you want to make sure you know exactly what's on the server you can rename the A3 directory to A3.day1 and then use

```
git clone https://gitlab.socs.uoguelph.ca/2750W23/skremer/A3
```

to get a copy of exactly what is currently in the repo.

You can use the same command to retrieve your code on a different machine or architecture. E.g. to move your code from your laptop development to the SoCS server for testing.

The Tables

Create the following tables (make sure to match the spelling and case of the table names, and column names exactly):

Elements

Column Name	Data Type	PRIMARY KEY	AUTOINCREMENT	NOT NULL	FOREIGN KEY
ELEMENT_NO	INTEGER			✓	
ELEMENT_CODE	VARCHAR(3)	✓		✓	
ELEMENT_NAME	VARCHAR(32)			✓	
COLOUR1	CHAR(6)			✓	
COLOUR2	CHAR(6)			✓	
COLOUR3	CHAR(6)			✓	
RADIUS	DECIMAL(3)			✓	

Atoms

Column Name	Data Type	PRIMARY KEY	AUTOINCREMENT	NOT NULL	FOREIGN KEY
ATOM_ID	INTEGER	✓	✓	✓	
ELEMENT_CODE	VARCHAR(3)			✓	Elements(ELEMENT_CODE)
X	DECIMAL(7,4)			✓	
Y	DECIMAL(7,4)			✓	
Z	DECIMAL(7,4)			✓	

Bonds

Column Name	Data Type	PRIMARY KEY	AUTOINCREMENT	NOT NULL	FOREIGN KEY
BOND_ID	INTEGER	✓	✓	✓	
A1	INTEGER			✓	
A2	INTEGER			✓	
EPAIRS	INTEGER			✓	

Molecules

Column Name	Data Type	PRIMARY KEY	AUTOINCREMENT	NOT NULL	FOREIGN KEY
MOLECULE_ID	INTEGER	✓	✓	✓	
NAME	TEXT	UNIQUE		✓	

MoleculeAtom

Column Name	Data Type	PRIMARY KEY	AUTOINCREMENT	NOT NULL	FOREIGN KEY
MOLECULE_ID	INTEGER	✓		✓	Molecules(MOLECULE_ID)
ATOM_ID	INTEGER	✓		✓	Atoms(ATOM_ID)

MoleculeBond

Column Name	Data Type	PRIMARY KEY	AUTOINCREMENT	NOT NULL	FOREIGN KEY
MOLECULE_ID	INTEGER	✓		✓	Molecules(MOLECULE_ID)
BOND_ID	INTEGER	✓		✓	Bonds(BOND_ID)

PART 0: Preparation

Make sure your `Molecule.parse` method does not `sort` the molecule. Move the `sort` method outside of the `parse` method as needed (e.g. in `do_POST`). If the `parse` method `sorts` your molecule, all your bonds will get confused.

Remove the `radius` and `element_name` dictionaries from MolDisplay.py.

The `molecule.append_bond` method in `molecule.i` subtracts `1` from the atom numbers because it assumes that the numbers being passed are coming from a `.sdf` file. In this assignment and the rest of the project, we will be reading from `.sdf` files, but more often, retrieving atom numbers from our database. In the database, the atom numbers will be numbered from zero (not one), so it makes more sense to move the `-1` operation from the `molecule.append_bond` method to the `Molecule.parse` method. Do this before you begin the assignment to make your life easier.

Remove `.decode('utf-8')` calls from your `parse` method so that you can read `.sdf` files rather than network traffic. I'll show you a way to restore the server functionality later.

Modify the output formatting string in the `svg` method of the `Atom` class from:

```
'  <circle cx="%.2f" cy="%.2f" r="%d" fill="%s"/>\n'
```

To:

```
'  <circle cx="%.2f" cy="%.2f" r="%d" fill="url(#%s)"/>\n'
```

PART I: Python code to put data into the database

In all the instructions provided below, be sure not to leave in any printing statements for debugging. Print only what you are told to print.

Create a `Database` class with the following methods:

`__init__( self, reset=False ):`
This constructor should create/open a database connection to a file in the local directory called "`molecules.db`" and store it as a class attribute. If `reset` is set to `True`, it should first delete the file "`molecules.db`" so that a fresh database is created upon connection.

`create_tables( self ):`
This method should create the tables described above. If any of the tables already exist, it should leave them alone and not re-create them.

`__setitem__( self, table, values ):`
This method should provide a method to use indexing (i.e. `[key]`) to set the values in the table named `table` based on the values in the tuple `values` (see example code, below).

`add_atom( self, molname, atom ):`
This method should add the attributes of the **`atom`** object (class **`MolDisplay.Atom`**) to the **`Atoms`** table, and add an entry into the **`MoleculeAtom`** table that links the named molecule to the atom entry in the **`Atoms`** table.

`add_bond( self, molname, bond ):`
This method should add the attributes of the **`bond`** object (class **`MolDisplay.Bond`**) to the **`Bonds`** table, and add an entry into the **`MoleculeBond`** table that links the named molecule to the atom entry in the **`Bonds`** table.

`add_molecule( self, name, fp ):`
This function should create a **`MolDisplay.Molecule`** object, call its **`parse`** method on **`fp`**, add an entry to the **`Molecules`** table and call **`add_atom`** and **`add_bond`** on the database for each atom and bond returned by the **`get_atom`** and **`get_bond`** methods of the molecule.

You can write additional helper functions and methods as required.

Testing. You can add the following text to the end of your file to turn you molsql.py library into an executable program.

```
if __name__ == "__main__":
    db = Database(reset=True);
    db.create_tables();

    db['Elements'] = ( 1, 'H', 'Hydrogen', 'FFFFFF', '050505', '020202', 25 );
    db['Elements'] = ( 6, 'C', 'Carbon',   '808080', '010101', '000000', 40 );
    db['Elements'] = ( 7, 'N', 'Nitrogen', '0000FF', '000005', '000002', 40 );
    db['Elements'] = ( 8, 'O', 'Oxygen',   'FF0000', '050000', '020000', 40 );

    fp = open( 'water-3D-structure-CT1000292221.sdf' );
    db.add_molecule( 'Water', fp );

    fp = open( 'caffeine-3D-structure-CT1001987571.sdf' );
    db.add_molecule( 'Caffeine', fp );

    fp = open( 'CID_31260.sdf' );
    db.add_molecule( 'Isopentanol', fp );

    # display tables
    print( db.conn.execute( "SELECT * FROM Elements;" ).fetchall() );
    print( db.conn.execute( "SELECT * FROM Molecules;" ).fetchall() );
    print( db.conn.execute( "SELECT * FROM Atoms;" ).fetchall() );
    print( db.conn.execute( "SELECT * FROM Bonds;" ).fetchall() );
    print( db.conn.execute( "SELECT * FROM MoleculeAtom;" ).fetchall() );
    print( db.conn.execute( "SELECT * FROM MoleculeBond;" ).fetchall() );
```

PART II: Python code to generate svg files from the database

Add more methods to the **`Database`** class in **`molsql.py`** as follows:

```
def load_mol( self, name ):
```

This method returns a **MolDisplay.Molecule** object initialized based on the molecule named name. It will retrieve all the atoms in the database associated with the named molecule and **append_atom** them to the **Molecule** object in order of increasing **ATOM_ID**. IMPORTANT: Do not use 3 SQL commands to do this. Instead use **JOIN**s to do this in one step. Similarly, it will retrieve all the bonds in the database associated with the named molecule and **append_bond** them to the **Molecule** object in order of increasing **BOND_ID**. Use a single SQL command to select all the bonds.

def radius(self):
This method returns a Python dictionary mapping **ELEMENT_CODE** values to **RADIUS** values based on the **Elements** table.

def element_name(self):
This method returns a Python dictionary mapping **ELEMENT_CODE** values to **ELEMENT_NAME** values based on the **Elements** table.

def radial_gradients(self):
This method returns a Python string consisting of multiple concatenations of the following string constant:

```
radialGradientSVG = """
  <radialGradient id="%s" cx="-50%%" cy="-50%%" r="220%%" fx="20%%" fy="20%%">
    <stop offset="0%%" stop-color="#%s"/>
    <stop offset="50%%" stop-color="#%s"/>
    <stop offset="100%%" stop-color="#%s"/>
  </radialGradient>""";
```

with the **%s** values replaced by **ELEMENT_NAME**, **COLOUR1**, **COLOUR2**, and **COLOUR3** as retrieved from the **Element** table.

Testing. You can add the following text to the end of your file to turn you molsql.py library into an executable program.

```
if __name__ == "__main__":
    db = Database(reset=False); # or use default

    MolDisplay.radius = db.radius();
    MolDisplay.element_name = db.element_name();
    MolDisplay.header += db.radial_gradients();

    for molecule in [ 'Water', 'Caffeine', 'Isopentanol' ]:
        mol = db.load_mol( molecule );
        mol.sort();
        fp = open( molecule + ".svg", "w" );
        fp.write( mol.svg() );
        fp.close();
```

Testing

For **this** assignment, I will **not** be testing your code with dangerous data. All `.sdf` files will be sensible and I will not try to inject a "`; DROP TABLES;`" line into your SQL code.

You are responsible for testing your code to make sure that it works as required. The CourseLink web-site will be updated with some test programs to get you started. However, we will use a different set of test programs to grade your code, so you need to make sure that your code performs according to the instructions above by writing more test code.

Your assignment will be tested on the SoCS servers. If you are developing in a different environment, you will need to allow yourself enough time to test and debug your code on the target machine. If your code works on one machine/environment but not another, you code is incorrect. Correct code will work consistently across all machines/environments. We will NOT test your code on YOUR machine/environment.

Ask Questions

The instructions above are intended to be as complete and clear as possible. However, it is YOUR responsibility to resolve any ambiguities or confusion about the instructions by asking questions in class, via the discussion forums, or by e-mailing the course e-mail.

Nightmare mode

Add ellipses to the ends of the far-away side of the bonds. Each elipse should be scaled and aligned so that the long axis of the ellipse is parallel to and the same length as the short end of the rectangle representing the faw-away side of the bond. Scale the other axis of the ellipse proportionally so that it has zero width if bond lies entirely in the x-y plane (i.e. both ends of the bond have the same z-value), and so that it has a width equal to the first axis if the bond is entirely in the z axis (i.e. coming out of the screen towards you).

Add shading to the rectangles and ellipse to make the bonds look like 3-d cylinders that go into the atom spheres. See **`Nightmare.svg`** on CourseLink.

Git

You must submit your files using **`git`** to the School's git server. Only code submitted to the server will be graded. Do ***not*** e-mail your assignment to the instructor. We will only grade one submission; we will only grade the last submission that you make to the server and apply any late penalty based on the last submission. So once your code is complete and you have tested it and you are ready to have it graded make sure to commit and push all of your changes to the server, and then do not make any more changes to the A3 files on the server.

Academic Integrity

Throughout the entire time that you are working on this assignment. You must not look at another student's code, nor allow your code to be accessible to any other student. You can share additional test cases (beyond those supplied by the instructor) or discuss what the correct outputs of the test programs should be, but do not share ANY code with your classmates.

Also, do your own work, do not hire someone to do the work for you.

Grading Rubric

__init__	1
create_tables	6
__setitem__	3
add_atom	3
add_bond	3
add_molecule	6
load_molecule	6
radius	1
element_name	1
radial_gradients	2
style	4
makefile	4
Total	40