

Beyond the plain text file format

How a relational database file can take us
beyond the plain text file format

ACS Spring 2016 San Diego
TJ O'Donnell
gNova, Inc.
tjo@gnova.com

Topics

- Text file formats
- Atoms, bonds, coordinates, properties, ...
- Relational database tables and SQL
- SQLite is a relational database in a single file
- Chemical and biological data
- Extensibility of SQL schemas
- Using SQLite files, programming

Molecular text file formats

Multiple formats commonly used

- Small molecules
- Proteins
- Nucleic acids
- Proprietary formats
- Chemical and biological data

Most can be interconverted – Problem solved!?

- OpenBabel supports 118 formats

[illegible][illegible]

SDF plain text

1-1

Cerius2 12120216093D 1 1.00000

Structure written by MMmdl.

39 41 0 0 0 0 0 0 0 0999 V2000

27.7051	22.0403	17.0243	C	0	0	0	0	0	0
26.4399	22.0976	16.4318	N	0	0	0	0	0	0
25.5381	21.4424	17.2831	C	0	0	0	0	0	0
26.2525	20.9753	18.3748	C	0	0	0	0	0	0
27.5943	21.3608	18.2218	C	0	0	0	0	0	0
24.0821	21.3670	17.1082	C	0	0	0	0	0	0
26.1324	22.6824	15.1634	C	0	0	0	0	0	0

...

1	2	1	0	0	0
1	5	2	0	0	0
1	19	1	0	0	0
2	3	1	0	0	0
2	7	1	0	0	0
3	4	2	0	0	0
3	6	1	0	0	0
4	5	1	0	0	0
4	24	1	0	0	0
5	25	1	0	0	0
6	8	2	0	0	0
6	12	1	0	0	0

...

M END

> <Name>

1-1

> <Family>

A.1

> <IC50_uM>

0.06

> <set>

1

\$\$\$\$

JSON structured file

```
{
  "name": "1-1",
  "charge": null,
  "atoms": [
    {
      "idx": 1,
      "symbol": "C",
      "atomic_number": 6,
      "charge": 0,
      "x": 27.7051,
      "y": 22.0403,
      "z": 17.0243
    },
    {
      "idx": 2,
      "symbol": "N",
      "atomic_number": 7,
      "charge": 0,
      "x": 26.4399,
      "y": 22.0976,
      "z": 16.4318
    },
    ...],
  "bonds": [
    [1,19,1],
    [1,2,1],
    [1,5,2],
    [2,3,1],
    [2,7,1],
    ...],
  "properties": {
    "Family": "A.1",
    "IC50_uM": 0.06,
    "set": 1
  }
}
```

CML/XML structured file

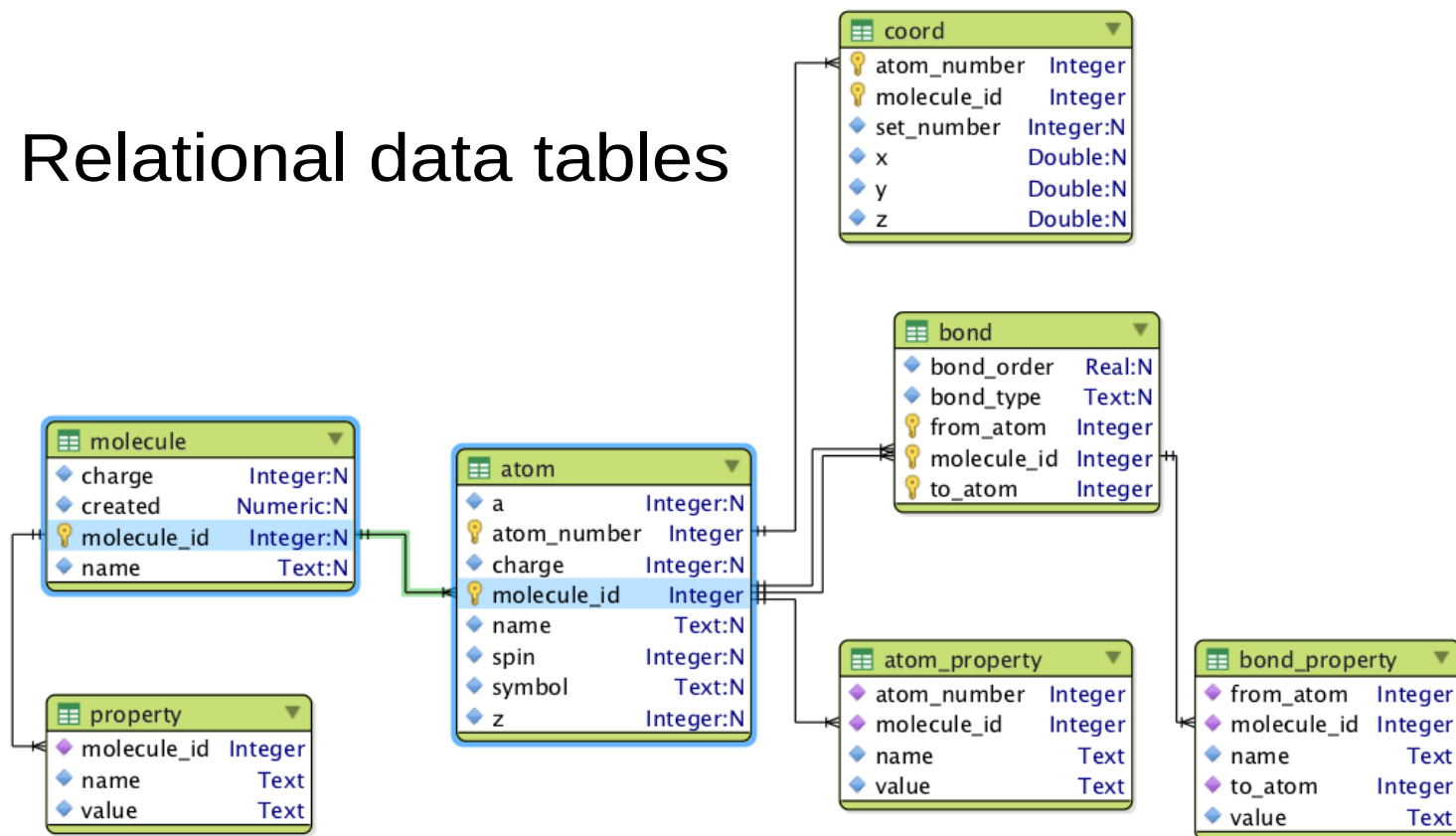
```
<?xml version="1.0"?>
<molecule id="id1-1" xmlns="http://www.xml-cml.org/schema">
  <atomArray>
    <atom id="a1" elementType="C" x3="27.705100" y3="22.040300" z3="17.024300" />
    <atom id="a2" elementType="N" x3="26.439900" y3="22.097600" z3="16.431800" />
    <atom id="a3" elementType="C" x3="25.538100" y3="21.442400" z3="17.283100" />
    <atom id="a4" elementType="C" x3="26.252500" y3="20.975300" z3="18.374800" />
    <atom id="a5" elementType="C" x3="27.594300" y3="21.360800" z3="18.221800" />
    <atom id="a6" elementType="C" x3="24.082100" y3="21.367000" z3="17.108200" />
    ...
  </atomArray>

  <bondArray>
    <bond atomRefs2="a1 a2" order="1" />
    <bond atomRefs2="a1 a5" order="2" />
    <bond atomRefs2="a1 a19" order="1" />
    <bond atomRefs2="a2 a3" order="1" />
    <bond atomRefs2="a2 a7" order="1" />
    <bond atomRefs2="a3 a4" order="2" />
    ...
  </bondArray>

  <propertyList>
    <property title="Name">
      <scalar>1-1</scalar>
    </property>
    <property title="Family">
      <scalar>A.1</scalar>
    </property>
    <property title="IC50_uM">
      <scalar>0.06</scalar>
    </property>
    <property title="set">
      <scalar>1</scalar>
    </property>
  </propertyList>
</molecule>
```

SQLite structured file .umdb

Relational data tables



What's in a SQLite file?

```
Select atom.atom_number, atom.symbol, coord.x, coord.y, coord.z, atom.charge
  From atom Join coord Using (molecule_id, atom_number)
 Where molecule_id=1 Order By atom.atom_number;
```

atom_number	symbol	x	y	z	charge
1	C	27.7051	22.0403	17.0243	0
2	N	26.4399	22.0976	16.4318	0
3	C	25.5381	21.4424	17.2831	0
4	C	26.2525	20.9753	18.3748	0
5	C	27.5943	21.3608	18.2218	0
6	C	24.0821	21.367	17.1082	0
7	C	26.1324	22.6824	15.1634	0
8	C	23.4105	22.2668	16.2675	0
9	C	22.022	22.2007	16.1197	0
10	C	21.2976	21.2409	16.8307	0
11	C	21.9509	20.3402	17.675	0
12	C	23.3399	20.4115	17.8175	0
13	C	26.3695	24.0457	14.9358	0
14	C	26.0627	24.6119	13.6959	0
15	C	25.5236	23.8179	12.691	0
16	C	25.2821	22.466	12.901	0
17	C	25.5848	21.8942	14.1391	0
18	F	25.2311	24.3643	11.5034	0
...					

Viewing and editing SQLite files

- Command line (built-in, all platforms)
- DB Browser for SQLite (open source on github)
- SQLite Manager Firefox addin
- SQLite Administrator (freeware, windows)
- Toad v4.2+ (not Mac)

SQLite Manager Firefox addin

Database Table Index View Trigger Tools Help

Directory (Select Profile Database) Go

Compound_000000001_000025000.umd

Structure Browse & Search Execute SQL DB Settings

Enter SQL

Select Data Manipulation

Run SQL Actions Last Error: not an error

molecule_id	name	value
12	MOL Chiral Flag	0
12	PUBCHEM_COMPOUND_CID	12
12	PUBCHEM_COMPOUND_CANONICALIZED	1
12	PUBCHEM_CACTVS_COMPLEXITY	104
12	PUBCHEM_CACTVS_HBOND_ACCEPTOR	4
12	PUBCHEM_CACTVS_HBOND_DONOR	4
12	PUBCHEM_CACTVS_ROTATABLE_BOND	0
12	PUBCHEM_CACTVS_SUBKEYS	AAADcYBgOAAAAAAAAAAAAAAAAAAAAAAAAAwAAAAAAAAABAAAGgAACAAACASAkAAwBoAAAgCAACBCAAACA...
12	PUBCHEM_IUPAC_OPENEYE_NAME	benzene-1,2,3,5-tetrol
12	PUBCHEM_IUPAC_CAS_NAME	benzene-1,2,3,5-tetrol
12	PUBCHEM_IUPAC_NAME	benzene-1,2,3,5-tetrol
12	PUBCHEM_IUPAC_SYSTEMATIC_NAME	benzene-1,2,3,5-tetrol
12	PUBCHEM_IUPAC_TRADITIONAL_NAME	benzene-1,2,3,5-tetrol
12	PUBCHEM_IUPAC_INCHI	InChI=1S/C6H6O4/c7-3-1-4(8)(10)5(9)2-3/h1-2,7-10H
12	PUBCHEM_IUPAC_INCHIKEY	RDJUHLPADHNP-UHFFFAOYSA-N
12	PUBCHEM_XLOGP3	0.8
12	PUBCHEM_EXACT_MASS	142.026609
12	PUBCHEM_MOLECULAR_FORMULA	C6H6O4
12	PUBCHEM_MOLECULAR_WEIGHT	142.10944
12	PUBCHEM_OPENEYE_CAN_SMILES	C1=C(C=C(C(=C1O)O)O)O
12	PUBCHEM_OPENEYE_ISO_SMILES	C1=C(C=C(C(=C1O)O)O)O
12	PUBCHEM_CACTVS_TPSA	80.9
12	PUBCHEM_MONOISOTOPIC_WEIGHT	142.026609
12	PUBCHEM_TOTAL_CHARGE	0
12	PUBCHEM_HEAVY_ATOM_COUNT	10

SQLite (www.sqlite.org)

- All the advantages of a flat-file; all the features of a relational database
- Self-contained, serverless, zero configuration
- Included in Python standard library
- Bindings for 43 languages
- Most widely deployed database engine in the world
- Public domain, free to use for any purpose

How large are SQLite files?

Original SDF contains 467 molecules

```
$ babel cox2_3d.sdf cox2_3d.pdb
$ babel cox2_3d.sdf cox2_3d.cml
$ python obmol.py cox2_3d.sdf cox2_3d.umdb
```

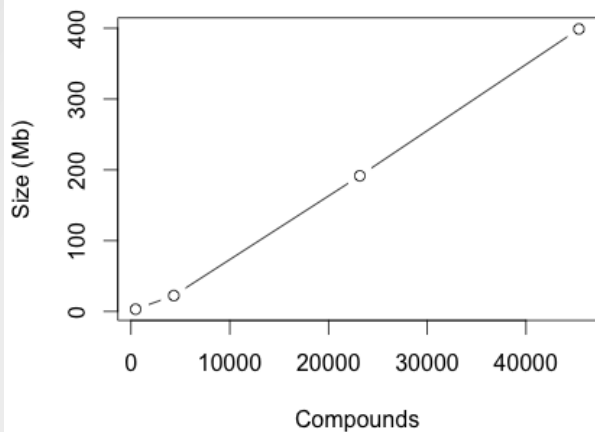
```
$ wc -c cox* | sort -n
1486991 cox2_3d.sdf
2419255 cox2_3d.cml
2444870 cox2_3d.js
2994176 cox2_3d.umdb
3115692 cox2_3d.pdb
```

```
$ gzip cox*; wc -c cox* | sort -n
247045 cox2_3d.sdf.gz (16%)
262544 cox2_3d.pdb.gz ( 8%)
273579 cox2_3d.js.gz (11%)
284060 cox2_3d.cml.gz (11%)
1293529 cox2_3d.umdb.gz (37%)
```

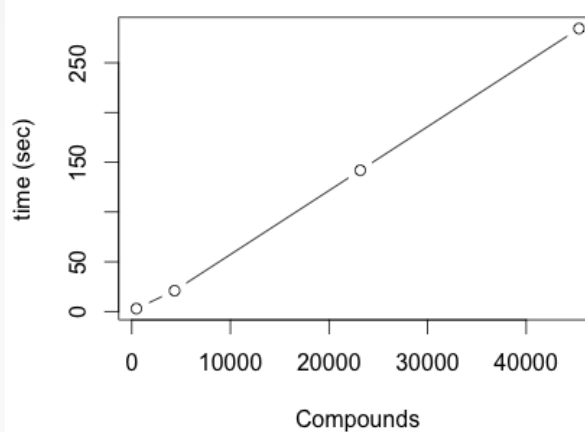
Maximum size allowed by filesystem

How do SQLite files scale?

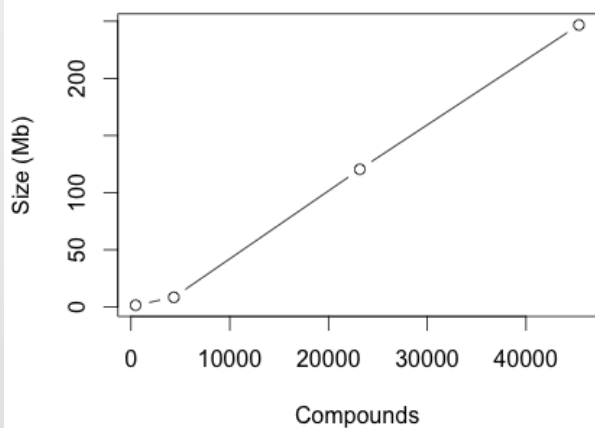
umdb size



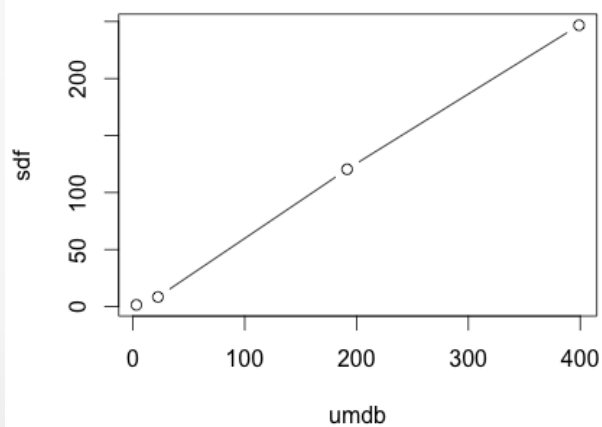
umdb create time



SDF size



File size



cox2	467
cas	4337
pubchem	23170
pubchem	45353

How efficient is using SQLite files?

Compute center of mass for 467 molecules

OpenBabel python vs. SQLite

```
$ time python cm.py
```

```
...  
14-15 397.4194432 24.5140548807 22.1725456909 15.9736236732  
14-16 412.88598 24.5963886245 22.3002506939 15.8502371715  
14-17 413.87404 24.5813303532 22.2904584772 15.8464177326  
7-39 328.3607232 23.7537457692 21.924262208 15.9951401499
```

```
real 0m0.842s  
user 0m0.811s  
sys 0m0.026s
```

```
$ time python cm-select.py
```

```
...  
14-15 397.4194432 24.5140548807 22.1725456909 15.9736236732  
14-16 412.88598 24.5963886245 22.3002506939 15.8502371715  
14-17 413.87404 24.5813303532 22.2904584772 15.8464177326  
7-39 328.3607232 23.7537457692 21.924262208 15.9951401499
```

```
real 0m0.166s  
user 0m0.136s  
sys 0m0.027s
```

SQLite is *fast* random access file

Any table can be indexed for fast access

Select data for first molecule

```
sqlite> select molecule_id,name,value from property
        where name in ('PUBCHEM_IUPAC_CAS_NAME', 'PUBCHEM_IUPAC_INCHI') And molecule_id=1;
```

```
molecule_id|name|value
1|PUBCHEM_IUPAC_CAS_NAME|3-acetyloxy-4-(trimethylammonio)butanoate
1|PUBCHEM_IUPAC_INCHI|InChI=1S/C9H17NO4/c1-7(11)14-8(5-9(12)13)6-10(2,3)4/h8H,5-6H2,1-4H3
Run Time: real 0.001 user 0.000293 sys 0.000087
```

Select data for molecule 20000

```
sqlite> select molecule_id,name,value from property
        where name in ('PUBCHEM_IUPAC_CAS_NAME', 'PUBCHEM_IUPAC_INCHI') And molecule_id=20000;
```

```
molecule_id|name|value
20000|PUBCHEM_IUPAC_CAS_NAME|2-thiocyanatoacetic acid butyl ester
20000|PUBCHEM_IUPAC_INCHI|InChI=1S/C7H11NO2S/c1-2-3-4-10-7(9)5-11-6-8/h2-5H2,1H3
Run Time: real 0.000 user 0.000279 sys 0.000089
```


Create SQLite .umdb files

Row by row, table by table taking data from anywhere

```
Insert into molecule (name, charge, created) Values ('aspirin', 0, datetime());  
Insert Into atom (molecule_id, atom_number, z) Values (1,1,6),(1,2,6),(1,3,7);  
Insert Into coord (molecule_id, atom_number, x, y, z) Values (1,1,0.0,0.0,0.0);  
...
```

467 molecules; python program using OpenBabel

```
$ time python obmol.py cox2_3d.sdf cox2_3d.umdb
```

```
real 0m2.854s  
user 0m2.252s  
sys 0m0.400s
```

Update SQLite .umdb files

```
sqlite> .timer on
sqlite> .mode csv
sqlite> .import newprop.csv newprop

sqlite> Insert Into property (molecule_id,name,value)
  Select molecule_id,'gnova_glogp',gnova_glogp From molecule
  Join newprop On (molecule.name=newprop.pubchem_id) ;
Run Time: real 0.807 user 0.283474 sys 0.469584

sqlite> Insert Into property (molecule_id,name,value)
  Select molecule_id,'gnova_tpsa',gnova_tpsa From molecule
  Join newprop On (molecule.name=newprop.pubchem_id) ;
Run Time: real 0.830 user 0.296428 sys 0.481289

sqlite> drop table newprop;
Run Time: real 0.038 user 0.006310 sys 0.021853

$ head newprop.csv
pubchem_id,gnova_glogp,gnova_tpsa
384,0.6933,86.63
385,1.1061,74.6
391,3.7655,132.91
399,-0.2772,111.9
409,1.8197,42.32
413,1.3615,41.99
416,-1.7999,149.65
420,7.3887,20.23
422,-3.3761,194.79
```

Using .umdb files in programs

- Compared to traditional object-oriented C++ or python
- Replacing object iterators with SQL select
- Replacing entire program with SQL computation

Structured data in programs

```
class OBAPI OBMol: public OBBase
{
protected:
    int                _flags;           //!< bitfield of flags
    bool               _autoPartialCharge; //!< Assign partial charges
    bool               _autoFormalCharge; //!< Assign formal charges
    std::string        _title;           //!< Molecule title
    std::vector<OBAtom*> _vatom;          //!< vector of atoms
    std::vector<OBAtom*> _atomIds;        //!< vector of atoms indexed by id
    std::vector<OBBond*> _vbond;          //!< vector of bonds
    std::vector<OBBond*> _bondIds;        //!< vector of bonds
    unsigned short int _dimension;       //!< Dimensionality of coordinates
    int                _totalCharge;     //!< Total charge on the molecule
    unsigned int        _totalSpin;      //!< Total spin on the molecule
    double              *_c;              //!< coordinate array
    std::vector<double*> _vconf;          //!< vector of conformers
    double              _energy;          //!< heat of formation
    unsigned int        _natoms;          //!< Number of atoms
    unsigned int        _nbonds;          //!< Number of bonds
    std::vector<OBResidue*> _residue;     //!< Residue information (if applicable)
    std::vector<OBInternalCoord*> _internals; //!< Internal Coordinates (if applicable)
    ...
}
```

Read text file into program objects

```
# compute center of mass

import openbabel

mol = openbabel.OBMol()
conv = openbabel.OBConversion()
conv.SetInFormat("sdf")

notatend = conv.ReadFile(mol,"cox2_3d.sdf")
while notatend:
    total_mass = 0.0
    cm_x = 0.0
    cm_y = 0.0
    cm_z = 0.0
    for atom in openbabel.OBMolAtomIter(mol):
        m = atom.GetAtomicMass()
        total_mass += m
        cm_x += m * atom.GetX()
        cm_y += m * atom.GetY()
        cm_z += m * atom.GetZ()

    print mol.GetTitle(), total_mass, \
        cm_x/total_mass, cm_y/total_mass, cm_z/total_mass
    mol.Clear()
    notatend = conv.Read(mol)
```

Read SQLite tables into program objects

Open the database file

Select all molecule, atoms, bonds, properties into program objects, corresponding to traditional reading of plain text file



```
Select molecule_id, molecule.name, atom.atom_number, atom.symbol, atom.charge, coord.x,
coord.y, coord.z, group_concat(to_atom) As bonds From molecule Join atom Using(molecule_id)
Join coord Using (molecule_id,atom_number) Join bond Using (molecule_id) Where
atom.atom_number=bond.from_atom Group by molecule_id, atom_number;
```

Iterate over objects to carry out program algorithms

```
total_mass = 0.0
cm_x = 0.0
cm_y = 0.0
cm_z = 0.0
for atom in openbabel.OBMolAtomIter(mol):
    m = atom.GetAtomicMass()
    total_mass += m
    cm_x += m * atom.GetX()
    cm_y += m * atom.GetY()
    cm_z += m * atom.GetZ()

print mol.GetTitle(), total_mass, mol.GetMolWt(), cm_x/total_mass,
cm_y/total_mass, cm_z/total_mass
```

Use SQL *select* as object *iterators*

Connect database file replaces file open and read

Database tables replace program objects

SQL *select* replaces object iterator

```
import sqlite3

conn = sqlite3.connect('cox2_3d.umdb')
conn.row_factory = sqlite3.Row
curs = conn.cursor()
curs.execute("Attach 'element.sqlite' As element")
sql = "Select molecule.name, mass, coord.x, coord.y, coord.z From molecule \
Join atom Using (molecule_id)\
Join element.element Using(z) \
Join coord Using (molecule_id, atom_number) \
Where molecule_id = ?"
for imol in range(467):
    total_mass = 0.0
    cm_x = 0.0
    cm_y = 0.0
    cm_z = 0.0
    for row in curs.execute(sql, [imol+1]):
        name = row['name']
        m = row['mass']
        total_mass += m
        cm_x += m * row['x']
        cm_y += m * row['y']
        cm_z += m * row['z']

    print name, total_mass, cm_x/total_mass, cm_y/total_mass, cm_z/total_mass
```

Using SQL for entire computation

Compute center of mass for 467 molecules using SQL

```
Attach "element.sqlite" As element;  
Select molecule.name, sum(mass),  
       sum(mass*coord.x)/sum(mass),  
       sum(mass*coord.y)/sum(mass),  
       sum(mass*coord.z)/sum(mass)  
From molecule Join atom Using (molecule_id)  
       Join element.element Using(z)  
       Join coord Using (molecule_id,atom_number)  
Group By molecule_id;
```

```
$ time sqlite3 cox2_3d.umdb <cm.sql
```

```
real    0m0.058s  
user    0m0.046s  
sys     0m0.008s
```

```
$ sqlite3 element.sqlite  
sqlite> select z,symbol,mass from element limit 10;
```

z	symbol	mass
1	H	1.00794
2	He	4.002602
3	Li	6.941
4	Be	9.012182
5	B	10.811
6	C	12.0107
7	N	14.0067
8	O	15.9994
9	F	18.9984032
10	Ne	20.1797

How efficient can SQLite be?

Compute center of mass for 467 molecules

OpenBabel python vs. SQLite

```
$ time python cm.py
```

```
real    0m0.842s
user    0m0.811s
sys     0m0.026s
```

```
$ time python cm-select.py
```

```
real    0m0.166s
user    0m0.136s
sys     0m0.027s
```

```
$ time sqlite3 cox2_3d.umdbs <cm.sql
```

```
real    0m0.059s
user    0m0.047s
sys     0m0.008s
```

Extending .umdb files

SQLite .umdb files can add new tables and relationships as well as update or add new data

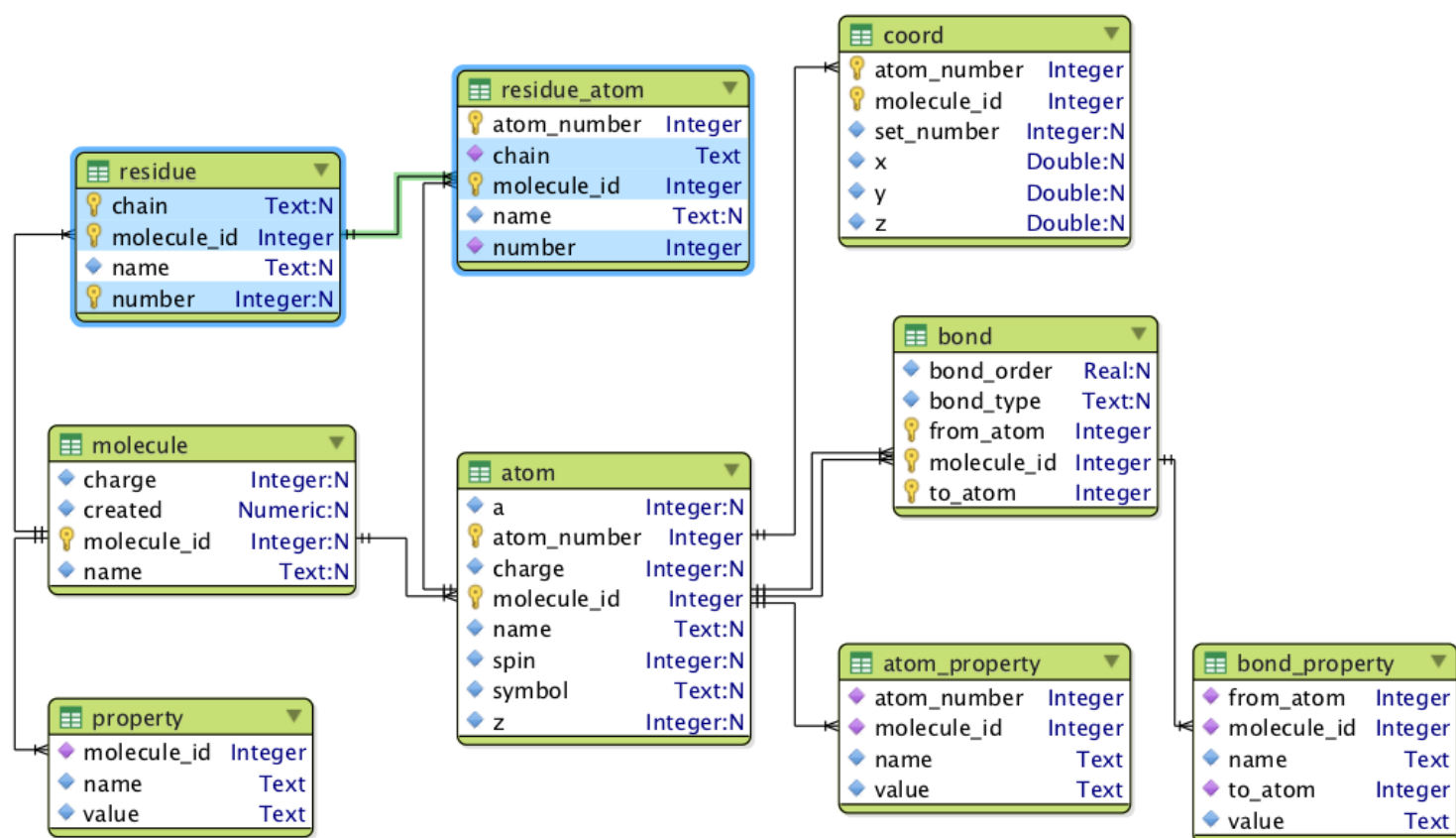
SD files v2000 limited to 999 atoms

- V3000 no atom limit, enhanced stereochemistry

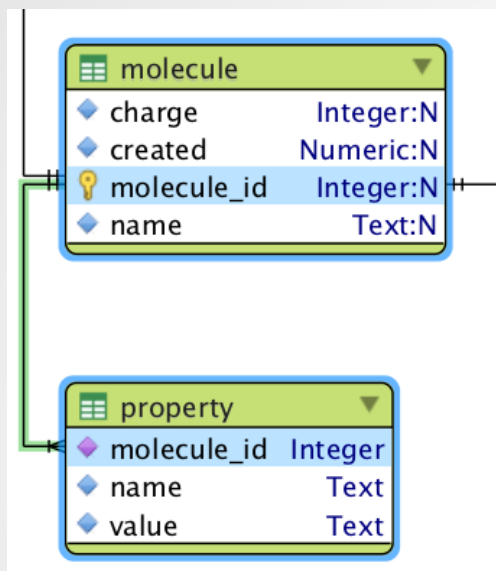
PDB files limited to 99999 atoms, 1-character chain, 3-character residue

- PDBML/XML, mmCIF eliminate column-width limits and add data definitions

Extended schema for proteins



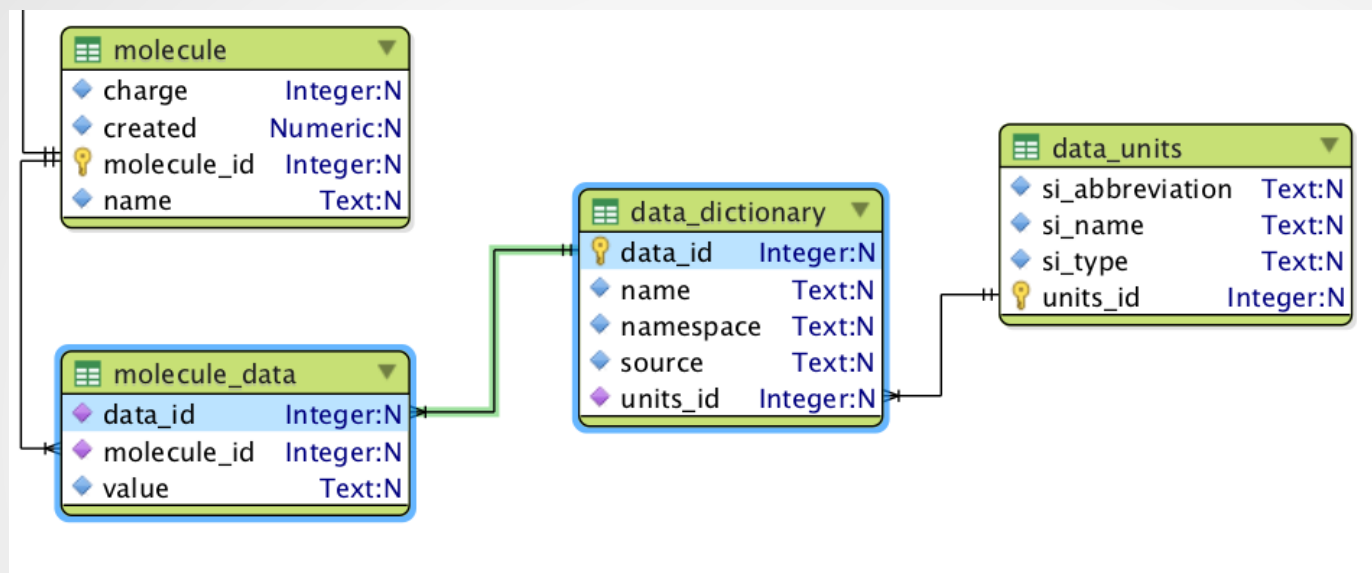
Chemical and biological data



```
Select property.name, property.value  
From property Where molecule_id=1;
```

Name	value
1-1	
Family	A.1
IC50_uM	0.06
Set	1

Extension for chemical/biological data



```
Select dict.name, data.value
From molecule_data As data Join data_dictionary dict Using (data_id)
Where molecule_id=1;
```

CML data dictionary

```
<?xml version="1.0" encoding="UTF-8" ?>
<dictionary xmlns="http://www.xml-cml.org/schema"
  xmlns:convention="http://www.xml-cml.org/convention/"
  xmlns:unit="http://www.xml-cml.org/unit/nonSi/"
  xmlns:unitType="http://www.xml-cml.org/unit/unitType/"
  xmlns:xhtml="http://www.w3.org/1999/xhtml"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  convention="convention:dictionary"
  title="fundamental chemistry concepts"
  namespace="http://www.xml-cml.org/dictionary/dummy/"
  dictionaryPrefix="dummy">
  <description>
    <xhtml:p>
      This is an example dictionary
    </xhtml:p>
  </description>

  <entry id="moleccmass" term="Molecular Mass"
    dataType="xsd:double" unitType="unitType:amount" units="unit:dalton">
    <definition>
      <xhtml:p>
        The mass of one mole of a substance in unified atomic mass units (Dalton).
      </xhtml:p>
    </definition>
    <description>
      <xhtml:p>
        The molecular mass (m) of a substance is the mass of one molecule of that substance,
        in unified atomic mass unit(s) ...
      </xhtml:p>
    </description>
  </entry>

  <entry id="molarmass" term="Molar Mass"
    dataType="xsd:double" unitType="unitType:amount" units="unit:dalton">
    <definition>
      <xhtml:p>
        The mass per amount of substance.
      </xhtml:p>
    </definition>
    <description>
      <xhtml:p>
        Molar mass, symbol M, is a physical property characteristic of a given substance
        ...
      </xhtml:p>
    </description>
  </entry>
</dictionary>
```

Summary

Advantages

- Single-File documents
- Cross-platform
- Fast access
- SQL
- Easily updated
- Readily extensible and reducible
- Concurrent use by multiple processes
- Transactions
- Better applications; restart files

Future work, other extensions

- umdb project at <http://github.com/tjod>
- PDBML/XML
 - “Third, the straightforward mapping of PDB data to relational database systems is retained.”
- mmCIF
 - flexible and extensible tag-value format for representing macromolecular structural data
- Extension tables
 - New types of data, for example spectroscopic data
 - Supramolecular assemblies