Assume that we wish to compute the average bytes read per second from an ASM diskgroup. Unfortunately there is no such simple way to do this because ASM instance views (*v$asm_disk*, *v$asm_disk_stats*, etc) keep those metrics as a cumulated value.

```
SQL> select to_char(sysdate,'DD/MM/YYYY HH24:MI:SS')
time,group_number,disk_number,reads,bytes_read from v$asm_disk order by group_number;

TIME                GROUP_NUMBER DISK_NUMBER      READS BYTES_READ
------------------- ------------ ----------- ---------- ----------
30/03/2009 18:17:00            1           0  203210936 2.1144E+13
30/03/2009 18:17:00            1           1  207653531 2.1099E+13
30/03/2009 18:17:00            1           7  200660160 2.1013E+13
30/03/2009 18:17:00            1           6  201889971 2.0885E+13
30/03/2009 18:17:00            1           5  200331993 2.0909E+13
30/03/2009 18:17:00            1           4  201634306 2.1150E+13
30/03/2009 18:17:00            1           3  205867346 2.1153E+13
30/03/2009 18:17:00            1           2  205056267 2.1232E+13
30/03/2009 18:17:00            2           0     950953 3.1857E+10
30/03/2009 18:17:00            2           6     331682 2.3616E+10
30/03/2009 18:17:00            2           5     161734 2.0774E+10
30/03/2009 18:17:00            2           4     158514 2.0783E+10
30/03/2009 18:17:00            2           3     250554 2.5873E+10
30/03/2009 18:17:00            2           2     159403 2.0792E+10
30/03/2009 18:17:00            2           1     230924 2.2017E+10
30/03/2009 18:17:00            2           7     244365 2.2155E+10

16 rows selected.

SQL> /

TIME                GROUP_NUMBER DISK_NUMBER      READS BYTES_READ
------------------- ------------ ----------- ---------- ----------
30/03/2009 18:17:18            1           0  203217604 2.1144E+13
30/03/2009 18:17:18            1           1  207659988 2.1099E+13
30/03/2009 18:17:18            1           7  200666316 2.1013E+13
30/03/2009 18:17:18            1           6  201896311 2.0886E+13
30/03/2009 18:17:18            1           5  200338148 2.0910E+13
30/03/2009 18:17:18            1           4  201639779 2.1150E+13
30/03/2009 18:17:18            1           3  205873000 2.1154E+13
30/03/2009 18:17:18            1           2  205062076 2.1233E+13
30/03/2009 18:17:18            2           0     950959 3.1857E+10
30/03/2009 18:17:18            2           6     331687 2.3616E+10
30/03/2009 18:17:18            2           5     161735 2.0774E+10
30/03/2009 18:17:18            2           4     158515 2.0783E+10
30/03/2009 18:17:18            2           3     250555 2.5873E+10
30/03/2009 18:17:18            2           2     159404 2.0792E+10
30/03/2009 18:17:18            2           1     230931 2.2017E+10
30/03/2009 18:17:18            2           7     244367 2.2155E+10

16 rows selected.

SQL> spool off
```

So in order to compute the rate of diskgroup read activity, a simple equation should be used. For example, if you want to compute average bytes read per second, you can use

$$Read\ Throughput = \frac{bytes\_read_2 - bytes\_read_1}{t_2 - t_1}$$

We will be looking for a possible implementation of this problem by using *SQLAlchemy* Python ORM tool.

Environment Setup

The very first thing is to include required libraries into your Python path. The two important ones you should do are Cx_Oracle and SQLAlchemy modules. Those are the Oracle connector for Python and

ORM module to be used (Although it will not be used explicitly *Cx_Oracle* will be internally used by *SQLAlchemy*)

Imports

Next step is to import necessary modules into your code so that you can access them. The necessary set of modules are given below

```
1:  #!/usr/bin/env python

2:  import time
3:  from sqlalchemy import create_engine
4:  from sqlalchemy.orm import sessionmaker
5:  from sqlalchemy.ext.declarative import declarative_base
6:  from sqlalchemy import Column,Integer,String
7:  from sqlalchemy.sql import func
```

Create an Engine and Session

Now we are ready to write our code. As in any kind of database access, the first thing to be defined is an access string to database. In *SQLAlchemy* the string format is pretty simple that

```
oracle://user:password@server:port/instance_name?[<optional parameters>]
```

One good thing about engine object is that if you set *echo* parameter to `True` in object creation, all SQL activities through this object will be logged on *stdout*.We will illustrate this later on.

The next thing we should do is to create a session class by using *sessionmaker* routine. Notice that we are using *engine* object in creating the session class. This corresponds to singleton pattern.

Final step is to instantiate session *s* from *Session* class.

```
 8:
 9:  engine = create_engine('oracle://sys:sysadm@localhost:1521/+ASM?mode=SYSDBA', echo=False)
10:  Session = sessionmaker(bind=engine)
11:  s = Session()
```

Declarative ORM Mapping

The magic behind ORM tools is may be the mapping of data objects to business objects. Once you have successfully mapped a relational database object to a Python object, all SQL stuff can be handled by ORM tool.

For this simple example, the approach will be the so-called *declarative base* mapping. In this approach only important points are

- You should carefully set *__table_name__* variable of the class to the correct table/view name in database
- You should match the table/view column names with the attribute names of the mapped class.
- You should choose one or more appropriate columns to form a primary key.

Then let *SQLAlchemy* handle rest for you. As you may find below we will be mapping *v$asm_disk* view to the Python class *Disk* extended from *Base* class. We have set *(group_number,disk_number)* tuple to form a primary key.

```
12:
13:  Base = declarative_base()
14:  class Disk(Base):
15:          __tablename__ = 'v$asm_disk'
16:
17:          group number = Column(Integer, primary key=True)
18:          disk_number = Column(Integer, primary_key=True)
```

```
19:          compound_index = Column(Integer)
20:          incarnation = Column(Integer)
21:          mount status = Column(String)
22:          header_status = Column(String)
23:          mode_status = Column(String)
24:          state = Column(String)
25:          redundancy = Column(String)
26:          library = Column(String)
27:          os_mb = Column(Integer)
28:          total_mb = Column(Integer)
29:          free_mb = Column(Integer)
30:          name = Column(String)
31:          failgroup = Column(String)
32:          label = Column(String)
33:          path = Column(String)
34:          udid = Column(String)
35:          product = Column(String)
36:          create_date = Column(String)
37:          mount_date = Column(String)
38:          repair_timer=Column(Integer)
39:          reads=Column(Integer)
40:          writes=Column(Integer)
41:          read_errs=Column(Integer)
42:          write_errs=Column(Integer)
43:          read_time=Column(Integer)
44:          write_time=Column(Integer)
45:          bytes_read=Column(Integer)
46:          bytes_written=Column(Integer)
47:          preferred_read=Column(String)
```

First Oracle Query without SQL

That's all!!! You can query your mapped view without writing a line of SQL. *SQLAlchemy* will generate the necessary SQL statements, issue them to database and return the result set for you. Let's see the *SQLAlchemy* way of writing

```
select * from v$asm_disk order by group_number
```

```
48:
49:  for x in  s.query(Disk).order_by(Disk.group_number):
50:          print x.group_number,x.disk_number,x.reads,x.bytes_read
```

A Complex One

Now we can write a more complex query to solve our problem. The query we should write is simply

```
select group_number,sum(bytes_read) as total_bytes_read from v$asm_disk group by group_number
order group_number
```

This can be written by *SQLAlchemy* as illustrated in line 58-61.

The only addition to above SQL statement is the *func.current_timestamp()* function on line 60. This function is used to find exact time on which our metric snapshot has taken. Later in the code this value will be used to compute time difference between successive snapshots.

Auxiliary *convert_to_second* function on line 52-53 is used to convert the time difference between two snapshots computed on line 68 from *timedelta* to seconds.

Rest of the code should be simple. The only built-in function I want to point out here is the *zip* function used on line 64 (I mention it because it was love at first sight for me). This function allows us to iterate over more than one array at a time. You can grasp what I mean by looking the usage within for loop.

```
51:
52:  def convert_to_second(tdelta):
53:          return tdelta.seconds + float(tdelta.microseconds)/1000000
```

```
54:
55:  current=[]
56:  previous=[]
57:  for sample in range(1,10):
58:          current = s.query(Disk.group_number,
59:                          func.sum(Disk.bytes_read).label('total_bytes_read'),
60:                          func.current_timestamp().label('t'))\
61:                          .group_by(Disk.group_number).order_by(Disk.group_number).all()
62:
63:          if sample>1:
64:                  for c,p in zip(current,previous):
65:                          print   'Diskgroup ' + repr(c.group_number) + \
66:                                  ' has and average read rate of ' + \
67:                                  repr((c.total_bytes_read - p.total_bytes_read)/ \
68:                                  convert_to_second(c.t-p.t)/1024) + ' kb/s'
69:                  print ''
70:
71:          previous = current
72:          time.sleep(5)
```

Joining Python Mappings/Oracle Tables

You should notice that since *v$asm_disk* view only has *group_number* column in it, we couldn't display the diskgroup name in output generated on line 65-68. The obvious solution is to join *v$asm_disk* view with *v$asm_diskgroup* over *group_number* column and then to do the group by operation over name column of *v$asm_diskgroup* view.

In order to access *v$asm_diskgroup* table we create another ORM mapping for *Diskgroup* class just like we did for *Disk* class.

```
73:
74:  class Diskgroup(Base):
75:          __tablename__ = 'v$asm_diskgroup'
76:
77:          group_number = Column(Integer, primary_key=True)
78:          name = Column(String)
79:          sector_size = Column(Integer)
80:          block_size = Column(Integer)
81:          allocation_unit_size = Column(Integer)
82:          state = Column(String)
83:          type = Column(String)
84:          total_mb = Column(Integer)
85:          free_mb = Column(Integer)
86:          required_mirror_free_mb = Column(Integer)
87:          usable_file_mb = Column(Integer)
88:          offline_disks = Column(Integer)
89:          compatibility = Column(String)
90:          database_compatibility = Column(String)
```

Then we do the join

```
 91:
 92:  current=[]
 93:  previous=[]
 94:  for sample in range(1,10):
 95:          current = s.query(Diskgroup.name,
 96:                          func.sum(Disk.bytes_read).label('total_bytes_read'),
 97:                          func.current_timestamp().label('t'))\
 98:                          .filter(Disk.group_number==Diskgroup.group_number)
 99:                          .group_by(Disk.group_number).order_by(Diskgroup.name).all()
100:
101:          if sample>1:
102:                  for c,p in zip(current,previous):
103:                          print   'Diskgroup ' + c.name + \
104:                                  ' has and average read rate of ' + \
105:                                  repr((c.total_bytes_read - p.total_bytes_read)/ \
106:                                  convert_to_second(c.t-p.t)/1024) + ' kb/s'
107:                  print ''
108:
```

```
109:          previous = current
110:          time.sleep(5)
```

Final Remarks

The final thing may be to show the effect of setting `echo=True` in creating *engine* object in the final code we have written:

```
echo=False  Diskgroup DATA has and average read rate of 187274.60077049638 kb/s
            Diskgroup FRA has and average read rate of 43.563252409496869 kb/s

            Diskgroup DATA has and average read rate of 182640.39800948848 kb/s
            Diskgroup FRA has and average read rate of 1526.8787512081021 kb/s

            Diskgroup DATA has and average read rate of 109506.65738836407 kb/s
            Diskgroup FRA has and average read rate of 37.123522527307543 kb/s

            Diskgroup DATA has and average read rate of 116599.03260586936 kb/s
            Diskgroup FRA has and average read rate of 21.133678441270451 kb/s

            Diskgroup DATA has and average read rate of 117966.28731548331 kb/s
            Diskgroup FRA has and average read rate of 28.011231725832289 kb/s

            Diskgroup DATA has and average read rate of 283361.38298393821 kb/s
            Diskgroup FRA has and average read rate of 24.898441590979296 kb/s

            Diskgroup DATA has and average read rate of 215301.81926216441 kb/s
            Diskgroup FRA has and average read rate of 40.336687225448721 kb/s

            Diskgroup DATA has and average read rate of 423985.65846416715 kb/s
            Diskgroup FRA has and average read rate of 28.010812173498969 kb/s

echo=True   2009-03-31 02:01:32,796 INFO sqlalchemy.engine.base.Engine.0x...e6f0 BEGIN
            2009-03-31 02:01:32,812 INFO sqlalchemy.engine.base.Engine.0x...e6f0 SELECT
            v$asm_diskgroup.name AS v$asm_diskgroup_name, sum(v$asm_disk.bytes_read) AS
            total_bytes_read, CURRENT_TIMESTAMP AS t
            FROM v$asm_diskgroup, v$asm_disk
            WHERE v$asm_disk.group_number = v$asm_diskgroup.group_number GROUP BY
            v$asm_diskgroup.name ORDER BY v$asm_diskgroup.name
            2009-03-31 02:01:32,812 INFO sqlalchemy.engine.base.Engine.0x...e6f0 {}
            2009-03-31 02:01:38,046 INFO sqlalchemy.engine.base.Engine.0x...e6f0 SELECT
            v$asm_diskgroup.name AS v$asm_diskgroup_name, sum(v$asm_disk.bytes_read) AS
            total_bytes_read, CURRENT_TIMESTAMP AS t
            FROM v$asm_diskgroup, v$asm_disk
            WHERE v$asm disk.group number = v$asm diskgroup.group number GROUP BY
            v$asm_diskgroup.name ORDER BY v$asm_diskgroup.name
            2009-03-31 02:01:38,046 INFO sqlalchemy.engine.base.Engine.0x...e6f0 {}
            Diskgroup DATA has and average read rate of 432000.74426161288 kb/s
            Diskgroup FRA has and average read rate of 42.835200744108626 kb/s

            2009-03-31 02:01:43,217 INFO sqlalchemy.engine.base.Engine.0x...e6f0 SELECT
            v$asm_diskgroup.name AS v$asm_diskgroup_name, sum(v$asm_disk.bytes_read) AS
            total_bytes_read, CURRENT_TIMESTAMP AS t
            FROM v$asm_diskgroup, v$asm_disk
            WHERE v$asm_disk.group_number = v$asm_diskgroup.group_number GROUP BY
            v$asm_diskgroup.name ORDER BY v$asm_diskgroup.name
            2009-03-31 02:01:43,217 INFO sqlalchemy.engine.base.Engine.0x...e6f0 {}
            Diskgroup DATA has and average read rate of 482049.41315017152 kb/s
            Diskgroup FRA has and average read rate of 17.022475083352099 kb/s

            2009-03-31 02:01:48,421 INFO sqlalchemy.engine.base.Engine.0x...e6f0 SELECT
            v$asm_diskgroup.name AS v$asm_diskgroup_name, sum(v$asm_disk.bytes_read) AS
            total_bytes_read, CURRENT_TIMESTAMP AS t
            FROM v$asm_diskgroup, v$asm_disk
            WHERE v$asm_disk.group_number = v$asm_diskgroup.group_number GROUP BY
            v$asm diskgroup.name ORDER BY v$asm diskgroup.name
            2009-03-31 02:01:48,421 INFO sqlalchemy.engine.base.Engine.0x...e6f0 {}
            Diskgroup DATA has and average read rate of 394290.03983635048 kb/s
            Diskgroup FRA has and average read rate of 15.371927632039094 kb/s

            2009-03-31 02:01:53,562 INFO sqlalchemy.engine.base.Engine.0x...e6f0 SELECT
            v$asm_diskgroup.name AS v$asm_diskgroup_name, sum(v$asm_disk.bytes_read) AS
            total_bytes_read, CURRENT_TIMESTAMP AS t
            FROM v$asm_diskgroup, v$asm_disk
            WHERE v$asm_disk.group_number = v$asm_diskgroup.group_number GROUP BY
            v$asm_diskgroup.name ORDER BY v$asm_diskgroup.name
```

```
2009-03-31 02:01:53,562 INFO sqlalchemy.engine.base.Engine.0x...e6f0 {}
Diskgroup DATA has and average read rate of 390695.72393789911 kb/s
Diskgroup FRA has and average read rate of 18.672505455386425 kb/s

2009-03-31 02:01:58,703 INFO sqlalchemy.engine.base.Engine.0x...e6f0 SELECT
v$asm_diskgroup.name AS v$asm_diskgroup_name, sum(v$asm_disk.bytes_read) AS
total_bytes_read, CURRENT_TIMESTAMP AS t
FROM v$asm diskgroup, v$asm disk
WHERE v$asm_disk.group_number = v$asm_diskgroup.group_number GROUP BY
v$asm_diskgroup.name ORDER BY v$asm_diskgroup.name
2009-03-31 02:01:58,703 INFO sqlalchemy.engine.base.Engine.0x...e6f0 {}
Diskgroup DATA has and average read rate of 388214.13616730121 kb/s
Diskgroup FRA has and average read rate of 28.006197927191273 kb/s

2009-03-31 02:02:03,890 INFO sqlalchemy.engine.base.Engine.0x...e6f0 SELECT
v$asm diskgroup.name AS v$asm diskgroup name, sum(v$asm disk.bytes read) AS
total_bytes_read, CURRENT_TIMESTAMP AS t
FROM v$asm_diskgroup, v$asm_disk
WHERE v$asm_disk.group_number = v$asm_diskgroup.group_number GROUP BY
v$asm diskgroup.name ORDER BY v$asm diskgroup.name
2009-03-31 02:02:03,890 INFO sqlalchemy.engine.base.Engine.0x...e6f0 {}
Diskgroup DATA has and average read rate of 251328.72885951088 kb/s
Diskgroup FRA has and average read rate of 21.588892360674063 kb/s

2009-03-31 02:02:09,030 INFO sqlalchemy.engine.base.Engine.0x...e6f0 SELECT
v$asm_diskgroup.name AS v$asm_diskgroup_name, sum(v$asm_disk.bytes_read) AS
total_bytes_read, CURRENT_TIMESTAMP AS t
FROM v$asm_diskgroup, v$asm_disk
WHERE v$asm_disk.group_number = v$asm_diskgroup.group_number GROUP BY
v$asm_diskgroup.name ORDER BY v$asm_diskgroup.name
2009-03-31 02:02:09,030 INFO sqlalchemy.engine.base.Engine.0x...e6f0 {}
Diskgroup DATA has and average read rate of 281795.95753356797 kb/s
Diskgroup FRA has and average read rate of 40.485002544910614 kb/s

2009-03-31 02:02:14,171 INFO sqlalchemy.engine.base.Engine.0x...e6f0 SELECT
v$asm_diskgroup.name AS v$asm_diskgroup_name, sum(v$asm_disk.bytes_read) AS
total_bytes_read, CURRENT_TIMESTAMP AS t
FROM v$asm_diskgroup, v$asm_disk
WHERE v$asm_disk.group_number = v$asm_diskgroup.group_number GROUP BY
v$asm_diskgroup.name ORDER BY v$asm_diskgroup.name
2009-03-31 02:02:14,171 INFO sqlalchemy.engine.base.Engine.0x...e6f0 {}
Diskgroup DATA has and average read rate of 289819.87225551077 kb/s
Diskgroup FRA has and average read rate of 18.674957310214772 kb/s
```

As you see when `echo=True`, you can see any SQL statements that ORM have generated and sent to Oracle database. This is critical for some complex SQL statements to be generated. You may want to ensure that ORM tool has generated the convenient SQL for Oracle.

Conclusion

As you may see, ORM tools allow you to clean up your code from SQL statements. Moreover you no longer need to think about database layer details. You can directly start working on business objects (mapping classes).

But keep in mind that ORM tools are also not silver bullets. They may ease your life up to 80%-90%. After that point you should again write your own SQL, PL/SQL statements to prepare the necessary data for business layer. Moreover, keep in mind that there is always a possibility of “bad” SQL generation by the ORM tools which may cause some SQL execution plan headaches.