所以什么是ORM?

```
O --> Object 对象
R --> Relation 关系
M --> Mapping 映射
```

在我们创建表之前我们要熟悉我们的ORM操作

1. Class --> Object
2. 创建数据库引擎
3. 将所有的Class序列化成数据表

什么是ORM模型?

```
"""
ORM模型 - obj 里面的属性 == table中创建的字段
        - obj 定义table的操作方式和属性
        这些都是Base帮我们决定的
"""
```

ok 那么我们现在就按照上面所说的步骤来，第一步：

首先我们先创建我们的class，我们所有的class都要继承我们的Base类，因为我们创建数据库中的字段之类的东西，都是Base帮我们创建的，ok，那我们如何引入Base呢？看下面的代码

```
from sqlalchemy.ext.declarative import declarative_base

# Base 是 ORM模型的基类
Base = declarative_base()
```

创建表

然后我们就要开始创建表了

```
from sqlalchemy import Column, Integer, String

# Column 是字段的意思 一列一列的那个
# Integer 代表的是数据类型
# primary_key 代表的是主键
# autoincrement 自增
# index 为索引

class User(Base): #这仅仅是一个类名，不是我们的表名
    __tablename__ = "user"
    id = Column(Integer,primary_key=True,autoincrement=True)
    name = Column(String(32), index=True)
```

ok 到了我们的第二步，创建数据库引擎：

我们要来指定需要创建在哪一个数据库里面，整个前提我们需要有这个数据库，不然是创建不成功的

```
from sqlalchemy import create_engine

engine = create_engine("mysql+pymysql://root:********@127.0.0.1:3306/sqlachemy?
charset=utf8")

# ****是你链接数据库的密码
```

ok 到了我们的第三步，就是将所有继承Base的class序列化成数据库

```
Base.metadata.create_all(engine)
```

ok 现在右键点击运行代码，就可以成功的将数据库表建出来了。

```
mysql> use sqlachemy;
Database changed
mysql> show tables;
Empty set (0.02 sec)

mysql> show tables;
+--------------------+
| Tables_in_sqlachemy |
+--------------------+
| user               |
```

```
+--------------------+
1 row in set (0.00 sec)

mysql> select * from user;
Empty set (0.00 sec)

mysql> desc user;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int         | NO   | PRI | NULL    | auto_increment |
| name  | varchar(32) | YES  | MUL | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
2 rows in set (0.01 sec)
```

增加数据

现在到了我们增加数据的操作，我们增加数据可以回想一下我们用原生sql的时候

```
"""
1.选中数据库 - 创建数据库引擎 导入数据库引擎
2.创建查询窗口 - 必须是选中数据库的查询窗口
3.创建sql语句
4.点击运行
"""
```

那么，上面这几个步骤对应着我们SQLAlchemy中要怎么写呢?

```
"""
1.选中数据库 - 创建数据库引擎 导入数据库引擎 -- engine =
create_engine("mysql+pymysql://root:newxin.001206@127.0.0.1:3306/sqlachemy?
charset=utf8")
2.创建查询窗口 - 必须是选中数据库的查询窗口
    from sqlalchemy.orm import sessionmaker
    Session_window = sessionmaker(engine)
3.创建sql语句
    user_obj = User(name="cyx")
4.点击运行
    db_session.add(user_obj)  # 将sql语句粘贴到窗口中
    db_session.commit()
"""
```

我们来具体的看一下代码的操作

- 第一步：需要先创建一个窗口

```
from sqlalchemy.orm import sessionmaker
Session_window = sessionmaker(engine)  # 询问你已经创建完查询窗口，是否打开
db_session = Session_window()  # 打开查询窗口
```

- 第二步：创建一个对象，并写上sql数据，准备运行

```
# 增加数据
# 首先我们需要创建一个对象
user_obj = User(name="cyx") # name="cyx" 就是增加一个name = cyx的一条数据
```

- 第三步，准备提交数据

```
db_session.add(user_obj)  # 将sql语句粘贴到窗口中
db_session.commit()
db_session.close()  # 关闭窗口
```

结果如下：

```
mysql> select * from user;
+----+------+
| id | name |
+----+------+
|  1 | cyx  |
+----+------+
1 row in set (0.00 sec)
```

插入多条数据

- 插入一条数据我们使用的是add(对象)
- 插入多条数据我们用的是add_all(对象列表)

具体的代码操作：

```
from sqlalchemy.orm import sessionmaker
Session_window = sessionmaker(engine)  # 询问你已经创建完查询窗口，是否打开

db_session = Session_window()  # 打开查询窗口
user_obj_list = [User(name="林丹"),User(name="谌龙")]

db_session.add_all(user_obj_list) # 注意这里使用的是add_all
db_session.commit()
db_session.close()  # 关闭窗口
```

ok 让我们看看数据库中的结果：

```
mysql> select * from user;
+----+--------+
| id | name   |
+----+--------+
|  1 | cyx    |
|  2 | 林丹   |
|  3 | 谌龙   |
+----+--------+
3 rows in set (0.00 sec)
```

查询数据

我们任何一个操作都需要创建一个查询窗口，上面已经提到过，我们应该如何创建一个查询窗口呢？

```
from sqlalchemy.orm import sessionmaker
# 创建查询窗口
session_window = sessionmaker(engine)
db_session = session_window()
```

查询第一条数据

我们现在开始查询数据

```
# 1. 查询数据
user = db_session.query(User).first()
print(user)

# query的中文意思有查询的意思
# 我们可以将语句理解为 在窗口中查询User这个表的第一条数据

# output
<__main__.User object at 0x00000187BF28D6A0>

# 可以很清楚的看见输出出来的而是一个对象，纳闷对象我们是可以点里面的属性的
```

取出数据

```
user = db_session.query(User).first()
print(user.id,user.name)

# output
1 cyx
```

查询多条数据

我们用的是.all()取出所有的数据

```
user_ogj_list = db_session.query(User).all()
for row in user_ogj_list:
    print(row.name)
# output
cyx
林丹
谌龙
```

带条件的查询

带条件的查询我们有两种形式：

- filter --> 这是以表达式的形式使用
- filter_by -- > 这是以传参的形式使用

我们大部分使用的是filter，filter的意思是“过滤”，因为我们写表达式比较熟悉，所以使用filter多一点，下面我们来看看具体的用法：

- filter(User.id <= 2) 查询 id<=2的数据

```
user_obj_list = db_session.query(User).filter(User.id <= 2).all() # 表达式的形式
for row in user_obj_list:
    print(row.name)

# output
cyx
林丹
```

- filter(User.id <= 2,User.name == "cyx") 在id <=2 和 name="cyx" 要同时成立

```
user_obj_list = db_session.query(User).filter(User.id <= 2,User.name ==
"cyx").all() # 表达式的形式 并且逗号隔开代表的是and条件
for row in user_obj_list:
    print(row.name)
```

- filter_by(id=1,name="cyx") 传参的形式

```
user_obj_list = db_session.query(User).filter_by(id=1,name="cyx").all()  # 传参的
形式
for row in user_obj_list:
    print(row.name)
```

更新数据

首先我们应该知道，每一个操作之前我们都要进行创建窗口（这里强调一万次）！怎么创建?

```
from sqlalchemy.orm import sessionmaker

session_windows = sessionmaker(engine)
db_session = session_windows()
```

更新一条数据

```
user_obj = db_session.query(User).filter(User.id==1).update({"name":"cyx123"})
db_session.commit()
```

ok我们看一下到底修改了没有

```
mysql> select * from user;
+----+--------+
| id | name   |
+----+--------+
|  1 | cyx123 |
|  2 | 林丹   |
|  3 | 谌龙   |
+----+--------+
3 rows in set (0.00 sec)
```

更新多条数据

```
user_obj = db_session.query(User).filter(User.id>1).update({"name":"666"})
db_session.commit()
```

ok 让我们看一下数据库

```
mysql> select * from user;
+----+--------+
| id | name   |
+----+--------+
|  2 | 666    |
|  3 | 666    |
|  1 | cyx123 |
+----+--------+
3 rows in set (0.00 sec)
```

删除数据

删除一条数据

```
# 1.删除一条数据
res = db_session.query(User).filter(User.id==1).delete()
db_session.commit()
```

看看数据库

```
mysql> select * from user;
+----+------+
| id | name |
+----+------+
|  2 | 666  |
|  3 | 666  |
+----+------+
2 rows in set (0.00 sec)
```

删除多条数据

```
res = db_session.query(User).filter(User.id>=1).delete()
db_session.commit()
```

看看数据库

```
mysql> select * from user;
Empty set (0.00 sec)
```

一对多（ForeignKey）

一对多我们的例子是学生与学校，学生只能有一个学校，一个学校能有多个学生

一开始的操作和一开始的一样，我们现在复习一下

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column,Integer,String,create_engine,ForeignKey
from sqlalchemy.orm import sessionmaker

Base = declarative_base() # 创建基类
engine = create_engine("mysql+pymysql://root:*****@127.0.0.1:3306/sqlachemy?
charset=utf8") # 创建引擎
```

我们现在开始建两张表，一张是sthdent，还有一个是school，但是这两张表与上面的创建有一点区别，区别在于ForeignKey的存在，这个是外键

```
class Student(Base):
    __tablename__= "student"
    id = Column(Integer,primary_key=True)
    name = Column(String(32))
    school_id = Column(Integer,ForeignKey("school.id"))  # 因为是要去找数据库里面表
的id，所以是表名不是类名
                                                          #外键关联到school的主键

class School(Base):
    __tablename__= "school"
    id = Column(Integer,primary_key=True)
    name = Column(String(32))

Base.metadata.create_all(engine)
```

大家看一下我上面的注释就明白了，我们现在开始创建表：

```
mysql> show tables;
+---------------------+
| Tables_in_sqlachemy |
+---------------------+
| school              |
| student             |
| user                |
+---------------------+
3 rows in set (0.04 sec)

mysql> desc student;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| id        | int         | NO   | PRI | NULL    | auto_increment |
| name      | varchar(32) | YES  |     | NULL    |                |
| school_id | int         | YES  | MUL | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> desc school;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int         | NO   | PRI | NULL    | auto_increment |
| name  | varchar(32) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
```

```
2 rows in set (0.00 sec)
```

但是我们现在还看不出有什么用处，因为我们还缺一点东西，所以我们来从添加数据开始引用：

添加数据

我们想一想，我们添加数据的时候，school_id怎么样才能对应到呢？

这里我们就要用到relationship的作用了，要让这两个ORM对象产生关系，那我们就要在ORM的层面上去建立关系，那么我们如何建表呢，我们一般在多的那一方建立关系，我们将上面的student表改成如下，只添加了一句话：

```
class Student(Base):
    __tablename__= "student"
    id = Column(Integer,primary_key=True)
    name = Column(String(32))
    school_id = Column(Integer,ForeignKey("school.id"))  # 因为是要去找数据库里面表的id，所以是表名不是类名  #外键关联到school的主键
    stu2sch = relationship("School",backref="sch2stu") # 不在数据库层面 而在ORM层面
```

那么现在我们就不需要自己去给school_id赋值，因为ORM会帮我们做这一切，这就是ORM的好用之处了，我们现在来学习一下添加数据的代码是怎么写的

正向添加数据

通过学生来添加学校

```
stu_obj = Student(name="cyx",stu2sch=School(name="南苑")) # id都是orm帮你实现
session = sessionmaker(engine) # 这里是创建窗口
db_session = session() # 这里是打开窗口

db_session.add(stu_obj)
db_session.commit()
```

然后我们看一下数据库的情况：

```
mysql> select * from student;
+----+------+-----------+
| id | name | school_id |
+----+------+-----------+
|  1 | cyx  |         1 |
+----+------+-----------+
1 row in set (0.00 sec)

mysql> select * from school;
+----+--------+
| id | name   |
+----+--------+
|  1 | 南苑   |
+----+--------+
1 row in set (0.00 sec)
```

反向添加数据

我们先将学校创建出来，然后创建学生，将学生的数据插入到学校内

具体的代码实现：

```
sch_obj = School(name="南苑2号")

student_list = [Student(name="张老板"),Student(name="张老板的小弟")]

sch_obj.sch2stu = student_list

db_session.add(sch_obj)
db_session.commit()
```

看看数据库里面是怎么实现的：

```
mysql> select * from school;
+----+------------+
| id | name       |
+----+------------+
|  1 | 南苑       |
|  2 | 南苑2号    |
+----+------------+
2 rows in set (0.00 sec)

mysql> select * from student;
+----+--------------------+-----------+
| id | name               | school_id |
+----+--------------------+-----------+
|  1 | cyx                |         1 |
|  2 | 张老板             |         2 |
|  3 | 张老板的小弟       |         2 |
+----+--------------------+-----------+
3 rows in set (0.00 sec)
```

查询数据

我们有了relationship我们查询数据就是非常的方便的，我们来看看

正向查询

```
stu_obj = db_session.query(Student).filter(Student.name=="张老板").first()
print(stu_obj.name,stu_obj.stu2sch.name)

# output
张老板 南苑2号
```

反向查询

我们是一对多，一个学校对应多个学校

```
sch_obj_list = db_session.query(School).all() # 首先将所有学校找到
for row in sch_obj_list: # 将每一个学校进行循环
    for stu in row.sch2stu: # 将每一个学校的学生进行循环
        print(row.name,stu.name) # 答应学校和学生的名字

# output
南苑 cyx
南苑2号 张老板
南苑2号 张老板的小弟
```

一对多的更新和删除数据是和单表操作是一样的

多对多

很快，我们就讲到多对多了，我对数据库的理解也就停留到了curd，最近打算入手MySQL的书来看看，看看里面的一些原理啊啥的，也许能让我们更加的理解MySQL吧，我也建议大家去学学，不过先把最最最最最基础的CURD学好吧。

首先我们使用SQLAlchemy第一步需要干嘛?

没错了，就是Base，还有一些建表必须使用的一些工具。

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, String, Integer, create_engine, ForeignKey
from sqlalchemy.orm import relationship

Base = declarative_base()

engine = create_engine(
    "mysql+pymysql://root:newxin.001206@127.0.0.1:3306/sqlachemy?charset=utf8",
    max_overflow=500,  # 超过连接池大小之外最多可以创建的链接
    pool_size=100,  # 连接池的大小
    echo=True  # 显示调试信息
)
```

ok，我们现在需要建立一个多对多的类型，我们就以男，女为多对多，第三张表就是酒店来建立他们的关系，这个时候我们就需要一个媒婆来给他们建立关系，这个媒婆就是secondary，这个东西来关联第三张表，具体怎么写呢?

```
class Girls(Base):
    __tablename__ = "girl"
    id = Column(Integer, primary_key=True)
    name = Column(String(32))
    g2b = relationship("Boys", backref="b2g", secondary="hotel")  # seconder是媒婆

class Boys(Base):
    __tablename__ = "boy"
    id = Column(Integer, primary_key=True)
    name = Column(String(32))

class Hotel(Base):
    """
    第三张表我们只想建立关系，关于relationship我们并不想放在这里
    原因是我们的relationship是通过boy找到girl，通过girl找到boy
```

```
    跟Hotel是你没有任何的关系
    """
    __tablename__ = "hotel"
    id = Column(Integer, primary_key=True)
    boy_id = Column(Integer, ForeignKey("boy.id")) # 这两行表示发生过关系的男女
    girl_id = Column(Integer, ForeignKey("girl.id"))

Base.metadata.create_all(engine) # 创建表
```

ok，那我们看看数据库建成什么样了，大家可以利用可视化的工具来看看，那样更加清楚，我这里就不用了。

```
mysql> desc hotel;
+---------+------+------+-----+---------+----------------+
| Field   | Type | Null | Key | Default | Extra          |
+---------+------+------+-----+---------+----------------+
| id      | int  | NO   | PRI | NULL    | auto_increment |
| boy_id  | int  | YES  | MUL | NULL    |                |
| girl_id | int  | YES  | MUL | NULL    |                |
+---------+------+------+-----+---------+----------------+
3 rows in set (0.33 sec)

mysql> desc boy;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int         | NO   | PRI | NULL    | auto_increment |
| name  | varchar(32) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql> desc girl;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int         | NO   | PRI | NULL    | auto_increment |
| name  | varchar(32) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)
```

我们现在要操作数据了，操作数据的第一步应该干什么？大家可以仔细想想

```
Session = sessionmaker(engine)
db_session = Session()
```

没错了，就是创建窗口。

添加数据

正向添加

这里的添加关系也是跟一张表的添加比较相似，第三张表的数据也是ORM自动帮我们创建了

```
girl_obj = Girls(name="zly")
girl_obj.g2b = [Boys(name="cyx")] # 因为是多对多的关系，所以要列表
db_session.add(girl_obj)
db_session.commit()
```

我们看看三张表创建成什么样了?

```
mysql> select * from girl;
+----+------+
| id | name |
+----+------+
|  1 | zly  |
+----+------+
1 row in set (0.00 sec)

mysql> select * from boy;
+----+------+
| id | name |
+----+------+
|  1 | cyx  |
+----+------+
1 row in set (0.00 sec)

mysql> select * from hotel;
+----+--------+---------+
| id | boy_id | girl_id |
+----+--------+---------+
|  1 |      1 |       1 |
+----+--------+---------+
1 row in set (0.00 sec)
```

反向添加

很容易理解

```
boy_obj = Boys(name="男一号")
boy_obj.b2g = [Girls(name="女一号"),Girls(name="女二号")]
db_session.add(boy_obj)
db_session.commit()
```

数据库中的数据

```
mysql> select * from girl;
+----+-----------+
| id | name      |
+----+-----------+
|  1 | zly       |
|  2 | 女一号    |
|  3 | 女二号    |
+----+-----------+
3 rows in set (0.00 sec)

mysql> select * from boy;
+----+-----------+
| id | name      |
```

```
+----+-----------+
|  1 | cyx       |
|  2 | 男一号    |
+----+-----------+
2 rows in set (0.00 sec)

mysql> select * from hotel;
+----+--------+---------+
| id | boy_id | girl_id |
+----+--------+---------+
|  1 |      1 |       1 |
|  2 |      2 |       2 |
|  3 |      2 |       3 |
+----+--------+---------+
3 rows in set (0.00 sec)
```

查询数据

正向

和一对多的查询数据差不多

```
girl_obj_list = db_session.query(Girls).all()
for girl in girl_obj_list:
    for boy in girl.g2b:
        print(girl.name, boy.name)

# output
zly cyx
女一号 男一号
女二号 男一号
```

反向

```
boy_obj_list = db_session.query(Boys).all()
for boy in boy_obj_list:
    for girl in boy.b2g:
        print(boy.name, girl.name)
```

一些高级操作

```
# 高级版查询操作,厉害了哦
#老规矩
from my_create_table import User,engine
from sqlalchemy.orm import sessionmaker

Session = sessionmaker(engine)
db_session = Session()

# 查询数据表操作
# and or
from sqlalchemy.sql import and_ , or_
ret = db_session.query(User).filter(and_(User.id > 3, User.name ==
'DragonFire')).all()
```

```
ret = db_session.query(User).filter(or_(User.id < 2, User.name ==
'DragonFire')).all()

# 查询数据 指定查询数据列 加入别名
r2 = db_session.query(User.name.label('username'), User.id).first()
print(r2.id,r2.username) # 15 NBDragon

# 原生SQL筛选条件
r4 = db_session.query(User).filter_by(name='DragonFire').all()
r5 = db_session.query(User).filter_by(name='DragonFire').first()

# 字符串匹配方式筛选条件 并使用 order_by进行排序
r6 = db_session.query(User).filter(text("id<:value and
name=:name")).params(value=224, name='DragonFire').order_by(User.id).all()

#原生SQL查询
r7 = db_session.query(User).from_statement(text("SELECT * FROM User where
name=:name")).params(name='DragonFire').all()

# 别名映射  name as nick
user_list = db_session.query(User.name.label("nick")).all()
print(user_list)
for row in user_list:
    print(row.nick) # 这里要写别名了

# 复杂查询
from sqlalchemy.sql import text
user_list = db_session.query(User).filter(text("id<:value and
name=:name")).params(value=3,name="DragonFire")

# 查询语句
from sqlalchemy.sql import text
user_list = db_session.query(User).filter(text("select * from User id<:value and
name=:name")).params(value=3,name="DragonFire")

# 排序 :
user_list = db_session.query(User).order_by(User.id).all()
user_list = db_session.query(User).order_by(User.id.desc()).all()
for row in user_list:
    print(row.name,row.id)

# 关闭连接
db_session.close()
```

高级修改数据

```
#高级版更新操作
from my_create_table import User,engine
from sqlalchemy.orm import sessionmaker

Session = sessionmaker(engine)
db_session = Session()

#直接修改
db_session.query(User).filter(User.id > 0).update({"name" : "099"})

#在原有值基础上添加 - 1
```

```
db_session.query(User).filter(User.id > 0).update({User.name: User.name + "099"},
synchronize_session=False)

#在原有值基础上添加 - 2
db_session.query(User).filter(User.id > 0).update({"age": User.age + 1},
synchronize_session="evaluate")
db_session.commit()
```