

## 18 | SQLAlchemy : 如何使用Python ORM框架来操作MySQL ?

2019-07-22 陈旸

SQL必知必会

[进入课程 >](#)



讲述：陈旸

时长 10:04 大小 13.85M



上节课，我介绍了 Python DB API 规范的作用，以及如何使用 MySQL 官方的 `mysql-connector` 驱动来完成数据库的连接和使用。在项目比较小的时候，我们可以直接使用 SQL 语句，通过 `mysql-connector` 完成与 MySQL 的交互，但是任何事物都有两面性，随着项目规模的增加，代码会越来越复杂，维护的成本也越来越高，这时 `mysql-connector` 就不够用了，我们需要更好的设计模式。

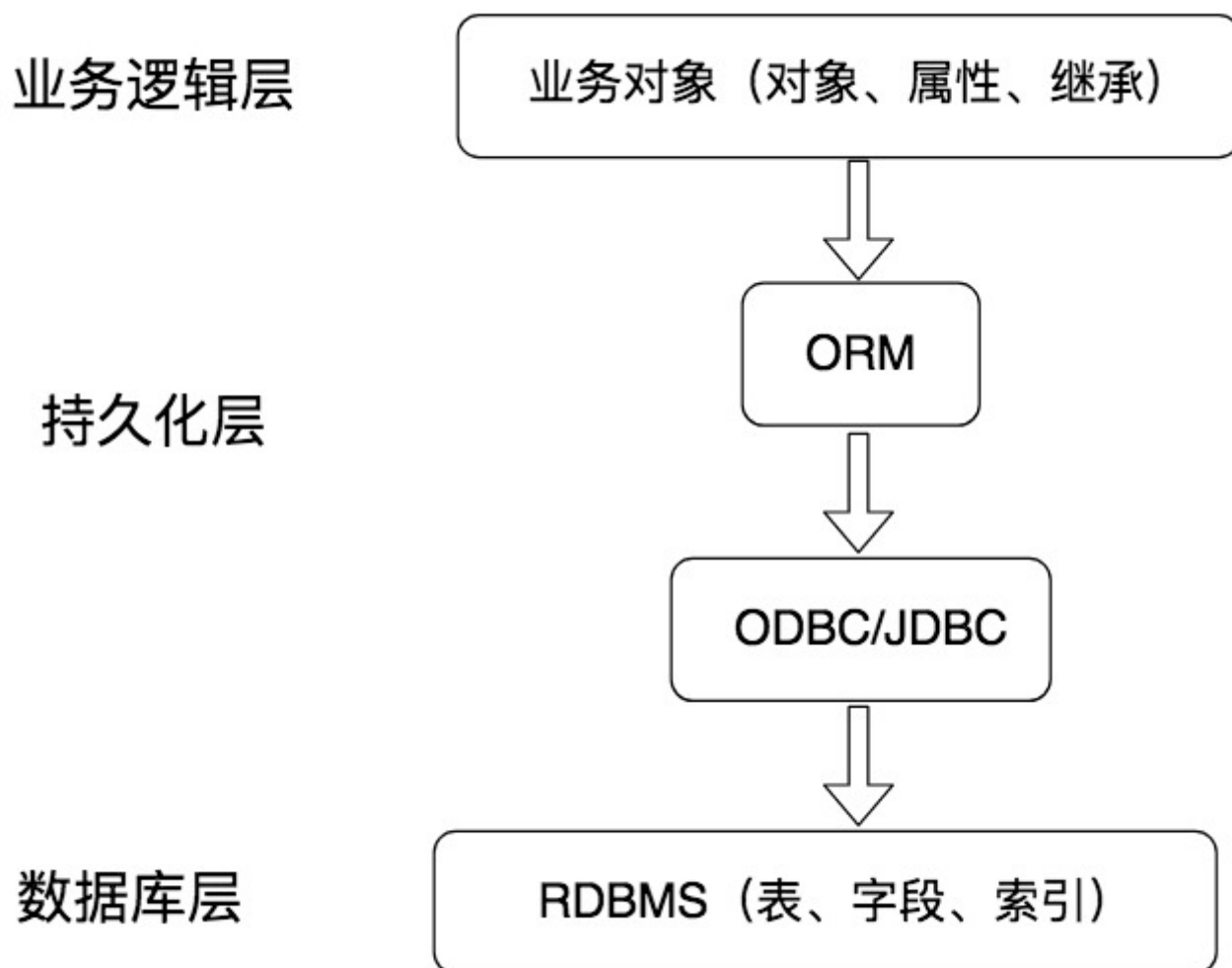
Python 还有另一种方式可以与 MySQL 进行交互，这种方式采用的是 ORM 框架。我们今天来讲解如何使用 ORM 框架操作 MySQL，那么今天的课程你需要掌握以下几个方面的内容：

1. 什么是 ORM 框架，以及为什么要使用 ORM 框架？

2. Python 中的 ORM 框架都有哪些？
3. 如何使用 SQLAlchemy 来完成与 MySQL 的交互？

## 我们为什么要使用 ORM 框架？

在讲解 ORM 框架之前，我们需要先了解什么是持久化。如下图所示，持久化层在业务逻辑层和数据库层起到了衔接的作用，它可以将内存中的数据模型转化为存储模型，或者将存储模型转化为内存中的数据模型。



你可能会想到，我们在讲事务的 4 大特性 ACID 时，提到过持久性。你可以简单地理解为，持久性就是将对象数据永久存储在数据库中。通常我们将数据库的作用理解为永久存储，将内存理解为暂时存储。我们在程序的层面操作数据，其实都是把数据放到内存中进行处理，如果需要数据就会通过持久化层，从数据库中取数据；如果需要保存数据，就是将对象数据通过持久化层存储到数据库中。

那么 ORM 解决的是什么问题呢？它提供了一种持久化模式，可以高效地对数据库进行访问。ORM 的英文是 Object Relation Mapping，中文叫对象关系映射。它是 RDBMS 和

业务实体对象之间的一个映射，从图中你也能看到，它可以把底层的 RDBMS 封装成业务实体对象，提供给业务逻辑层使用。程序员往往关注业务逻辑层面，而不是底层数据库该如何访问，以及如何编写 SQL 语句获取数据等等。采用 ORM，就可以从数据库的设计层面转化成面向对象的思维。

我在开篇的时候提到过，随着项目规模的增大，在代码层编写 SQL 语句访问数据库会降低开发效率，也会提升维护成本，因此越来越多的开发人员会采用基于 ORM 的方式来操作数据库。这样做的好处就是一旦定义好了对象模型，就可以让它们简单可复用，从而不必关注底层的数据库访问细节，我们只要将注意力集中到业务逻辑层面就可以了。由此还可以带来另一点好处，那就是即便数据库本身进行了更换，在业务逻辑代码上也不会有大的调整。这是因为 ORM 抽象了数据的存取，同时也兼容多种 DBMS，我们不用关心底层采用的到底是哪种 DBMS，是 MySQL，SQL Server，PostgreSQL 还是 SQLite。

但没有一种模式是完美的，采用 ORM 当然也会付出一些代价，比如性能上的一些损失。面对一些复杂的数据查询，ORM 会显得力不从心。虽然可以实现功能，但相比于直接编写 SQL 查询语句来说，ORM 需要编写的代码量和花费的时间会比较多，这种情况下，直接编写 SQL 反而会更简单有效。

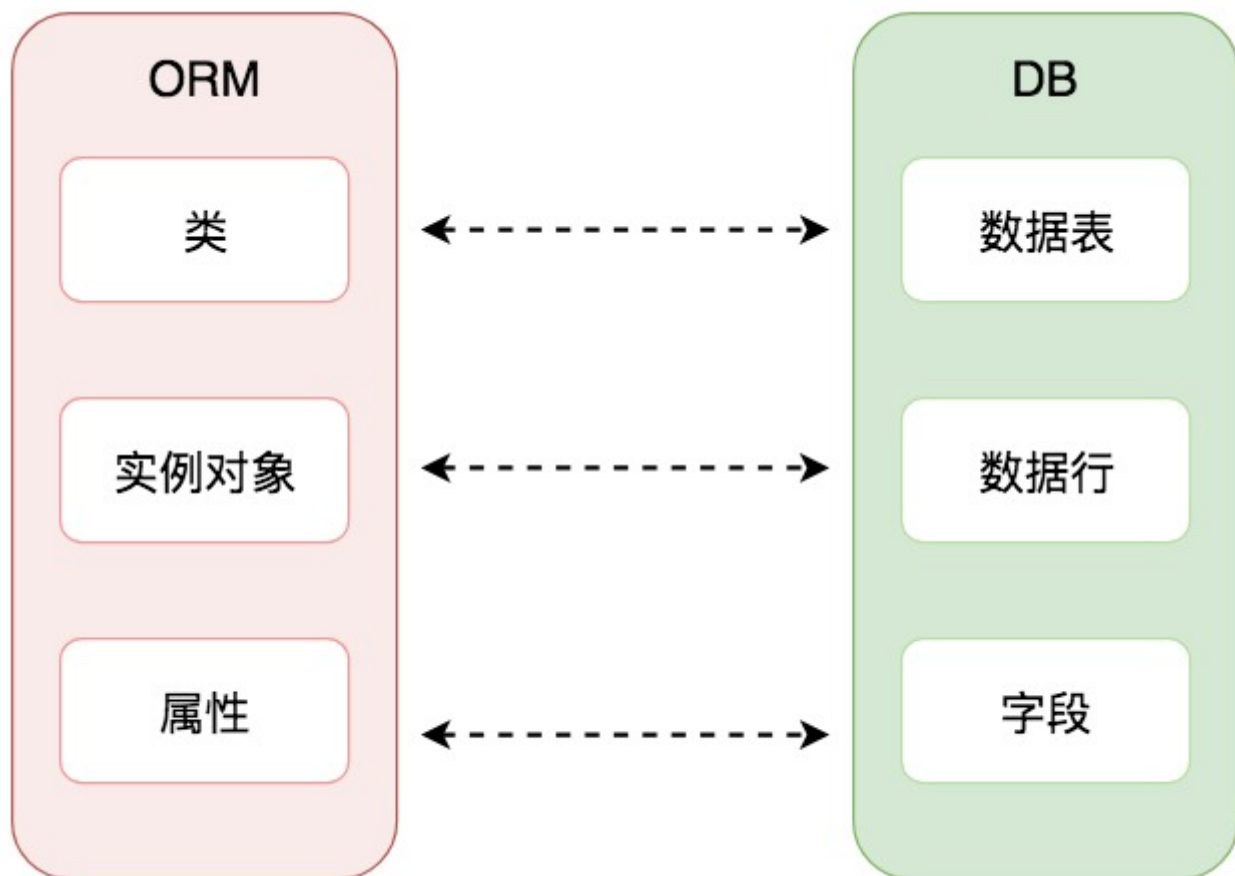
其实你也能看出来，没有一种方式是一劳永逸的，在实际工作中，我们需要根据需求选择适合的方式。

## Python 中的 ORM 框架都有哪些

ORM 框架帮我们适配了各种 DBMS，同时我们也可以选择不同的 ORM 框架。如果你用 Python 的话，有三种主流的 ORM 框架。

第一个是 Django，它是 Python 的 WEB 应用开发框架，本身走大而全的方式。Django 采用了 MTV 的框架模式，包括了 Model（模型），View（视图）和 Template（模版）。Model 模型只是 Django 的一部分功能，我们可以通过它来实现数据库的增删改查操作。

一个 Model 映射到一个数据表，如下图所示：



从这张图上你能直观地看到，ORM 的作用就是建立了对象关系映射。模型的每个属性代表数据表中的一个字段，我们通过操作类实例对象，对数据表中的数据行进行增删改查等操作。

第二个是 SQLAlchemy，它也是 Python 中常用的 ORM 框架之一。它提供了 SQL 工具包及 ORM 工具，如果你想用支持 ORM 和支持原生 SQL 两种方式的工具，那么 SQLAlchemy 是很好的选择。另外 SQLAlchemy 的社区更加活跃，这对项目实施会很有帮助。

第三个是 peewee，这是一个轻量级的 ORM 框架，简单易用。peewee 采用了 Model 类、Field 实例和 Model 实例来与数据库建立映射关系，从而完成面向对象的管理方式。使用起来方便，学习成本也低。

## 如何使用 SQLAlchemy 来操作 MySQL

下面我们来看下如何使用 SQLAlchemy 工具对 player 数据表进行增删改查，在使用前，你需要先安装相应的工具包：

```
1 pip install sqlalchemy
2 初始化数据库连接
3 from sqlalchemy import create_engine
4 # 初始化数据库连接，修改为你的数据库用户名和密码
5 engine = create_engine('mysql+mysqlconnector://root:password@localhost:3306/wucaai')
```

`create_engine` 的使用方法类似我们在上篇文章中提到的 `mysql.connector`，都需要提供数据库 + 数据库连接框架，即对应的是 `mysql+mysqlconnector`，后面的是用户名:密码@IP地址:端口号/数据库名称。

## 创建模型

我们已经创建了 `player` 数据表，这里需要创建相应的 `player` 模型。

 复制代码

```
1 # 定义 Player 对象:
2 class Player(Base):
3     # 表的名称:
4     __tablename__ = 'player'
5
6     # 表的结构:
7     player_id = Column(Integer, primary_key=True, autoincrement=True)
8     team_id = Column(Integer)
9     player_name = Column(String(255))
10    height = Column(Float(3,2))
```


这里需要说明的是，`__tablename__` 指明了模型对应的数据表名称，即 `player` 数据表。同时我们在 `Player` 模型中对采用的变量名进行定义，变量名需要和数据表中的字段名称保持一致，否则会找不到数据表中的字段。在 `SQLAlchemy` 中，我们采用 `Column` 对字段进行定义，常用的数据类型如下：

Integer	整数型
Float	浮点类型
Decimal	定点类型
Boolean	布尔类型
Date	datetime.date 日期类型
Time	datetime.time 时间类型
String	字符类型，使用时需要指定长度，区别于Text类型
Text	文本类型

除了指定 Column 的数据类型以外，我们也可以指定 Column 的参数，这些参数可以帮我们对对象创建列约束：

default	默认值
primary_key	是否为主键
unique	是否唯一
autoincrement	是否自动增长


这里需要说明的是，如果你使用相应的数据类型，那么需要提前在 SQLAlchemy 中进行引用，比如：

 复制代码

```
1 from sqlalchemy import Column, String, Integer, Float
```

### 对数据表进行增删改查

假设我们想给 player 表增加一名新球员，姓名为“约翰·科林斯”，球队 ID 为 1003（即亚特兰大老鹰），身高为 2.08。代码如下：

 复制代码

```
1 # 创建 DBSession 类型：
2 DBSession = sessionmaker(bind=engine)
3 # 创建 session 对象：
4 session = DBSession()
```

```

5
6
7 # 创建 Player 对象:
8 new_player = Player(team_id = 1003, player_name = " 约翰 - 科林斯 ", height = 2.08)
9 # 添加到 session:
10 session.add(new_player)
11 # 提交即保存到数据库:
12 session.commit()
13 # 关闭 session:
14 session.close()


```

这里，我们首先需要初始化 DBSession，相当于创建一个数据库的会话实例 session。通过 session 来完成新球员的添加。对于新球员的数据，我们可以通过 Player 类来完成创建，在参数中指定相应的 team\_id, player\_name, height 即可。

然后把创建好的对象 new\_player 添加到 session 中，提交到数据库即可完成添加数据的操作。

接着，我们来看一下如何查询数据。

添加完插入的新球员之后，我们可以查询下身高  $\geq 2.08\text{m}$  的球员都有哪些，代码如下：

 复制代码

```

1 # 增加 to_dict() 方法到 Base 类中
2 def to_dict(self):
3     return {c.name: getattr(self, c.name, None)
4             for c in self.__table__.columns}
5 # 将对象可以转化为 dict 类型
6 Base.to_dict = to_dict
7 # 查询身高 >=2.08 的球员有哪些
8 rows = session.query(Player).filter(Player.height >= 2.08).all()
9 print([row.to_dict() for row in rows])

```

运行结果：

 复制代码

```

1 [{'player_id': 10003, 'team_id': 1001, 'player_name': '安德烈 - 德拉蒙德', 'height': Deci


```



如果我们对整个数据行进行查询，采用的是`session.query(Player)`，相当于使用的是`SELECT *`。这时如果我们要在 Python 中对 query 结果进行打印，可以对 Base 类增加`to_dict()`方法，相当于将对象转化成了 Python 的字典类型。


在进行查询的时候，我们使用的是 filter 方法，对应的是 SQL 中的 WHERE 条件查询。除此之外，filter 也支持多条件查询。

如果是 AND 的关系，比如我们想要查询身高  $\geq 2.08$ ，同时身高  $\leq 2.10$  的球员，可以写成下面这样：

 复制代码

```
1 rows = session.query(Player).filter(Player.height >=2.08, Player.height <=2.10).all()
```

如果是 OR 的关系，比如我们想要查询身高  $\geq 2.08$ ，或者身高  $\leq 2.10$  的球员，可以写成这样：


 复制代码

```
1 rows = session.query(Player).filter(or_(Player.height >=2.08, Player.height <=2.10)).all()
```

这里我们使用了 SQLAlchemy 的 `or_` 操作符，在使用它之前你需要进行引入，即：`from sqlalchemy import or_`。

除了多条件查询，SQLAlchemy 也同样支持分组操作、排序和返回指定数量的结果。

比如我想要按照 `team_id` 进行分组，同时筛选分组后数据行数大于 5 的分组，并且按照分组后数据行数递增的顺序进行排序，显示 `team_id` 字段，以及每个分组的数据行数。那么代码如下：


 复制代码

```
1 from sqlalchemy import func
2 rows = session.query(Player.team_id, func.count(Player.player_id)).group_by(Player.team_id)
```



```
3 print(rows)
```

运行结果：

 复制代码

```
1 [(1001, 20), (1002, 17)]
```

这里有几点需要注意：

1. 我们把需要显示的字段 `Player.team_id`, `func.count(Player.player_id)` 作为 query 的参数，其中我们需要用到 sqlalchemy 的 `func` 类，它提供了各种聚集函数，比如 `func.count` 函数。
2. 在 `query()` 后面使用了 `group_by()` 进行分组，参数设置为 `Player.team_id` 字段，再使用 `having` 对分组条件进行筛选，参数为 `func.count(Player.player_id)>5`。
3. 使用 `order_by` 进行排序，参数为 `func.count(Player.player_id).asc()`，也就是按照分组后的数据行数递增的顺序进行排序，最后使用 `all()` 方法需要返回全部的数据。

你能看到 SQLAlchemy 使用的规则和使用 SELECT 语句的规则差不多，只是封装到了类中作为方法进行调用。

接着，我们再来看下如何删除数据。如果我们想要删除某些数据，需要先进行查询，然后再从 session 中把这些数据删除掉。


比如我们想要删除姓名为约翰·科林斯的球员，首先我们需要进行查询，然后从 session 对象中进行删除，最后进行 commit 提交，代码如下：

 复制代码

```
1 row = session.query(Player).filter(Player.player_name=='约翰 - 科林斯').first()
2 session.delete(row)
3 session.commit()
4 session.close()
```

需要说明的是，判断球员姓名是否为约翰·科林斯，这里需要使用（==）。

同样，如果我们想要修改某条数据，也需要进行查询，然后再进行修改。比如我想把球员索恩·马克的身高改成 2.17，那么执行完之后直接对 session 对象进行 commit 操作，代码如下：

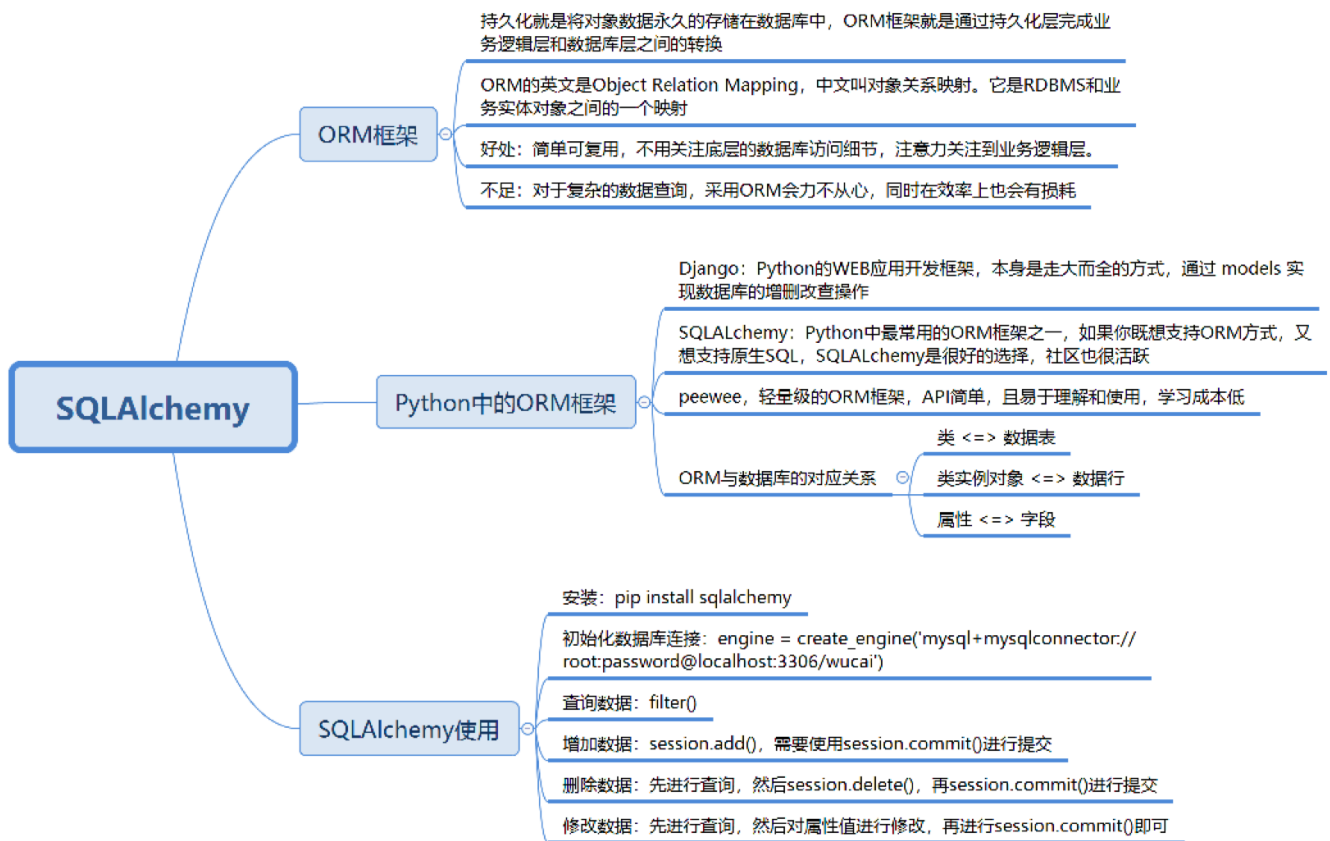
 复制代码

```
1 row = session.query(Player).filter(Player.player_name=='索恩 - 马克').first()
2 row.height = 2.17
3 session.commit()
4 session.close()
```

## 总结

今天我们使用 SQLAlchemy 对 MySQL 进行了操作，你能看到这些实现并不复杂，只是需要事先掌握一些使用方法，尤其是如何创建 session 对象，以及如何通过 session 对象来完成对数据的增删改查等操作。建议你把文章里的代码都跑一遍，在运行的过程中一定会有更深入的体会。

当然除了学习掌握 SQLAlchemy 这个 Python ORM 工具以外，我还希望你能了解到 ORM 的价值和不足。如果项目本身不大，那么自己动手写 SQL 语句会比较简单，你可以不使用 ORM 工具，而是直接使用上节课讲到的 mysql-connector。但是随着项目代码量的增加，为了在业务逻辑层与数据库底层进行松耦合，采用 ORM 框架是更加适合的。



我今天讲解了 SQLAlchemy 工具的使用，为了更好地让你理解，我出一道练习题吧。还是针对 player 数据表，请你使用 SQLAlchemy 工具查询身高为 2.08 米的球员，并且将这些球员的身高修改为 2.09。

欢迎你在评论区写下你的答案，也欢迎把这篇文章分享给你的朋友或者同事，一起交流。

# SQL 必知必会

从入门到数据实战

陈旻

清华大学计算机博士



新版升级：点击「👤 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 17 | 如何使用Python操作MySQL？

下一篇 19 | 基础篇总结：如何理解查询优化、通配符以及存储过程？

## 精选留言 (16)

写留言



ttttt

2019-07-22

缺少一些代码，可以参考廖雪峰的这个。

<https://www.liaoxuefeng.com/wiki/1016959663602400/1017803857459008>

展开 ∨



6



墨禾

2019-07-23

以下从ORM的作用，是什么，优缺点以及一些比较流行的ORM的对比的个人总结：

1.ORM的作用

对象关系映射，能够直接将数据库对象进行持久化。

...

展开 ▾



3



ttttt

2019-07-22

错误解决：

如果报如下错误：Authentication plugin 'caching\_sha2\_password' is not supported  
sqlalchemy.exc.NotSupportedError: (mysql.connector.errors.NotSupportedError)  
Authentication plugin 'caching\_sha2\_password' is not supported (Background on  
this error at: <http://sqlalche.me/e/tw8g>)...

展开 ▾



3



一叶知秋

2019-07-22

日常交作业~~~

# -\*- coding:utf-8 -\*-

from sqlalchemy import and\_

from sqlalchemy import Column, INT, FLOAT, VARCHAR...

展开 ▾



3



ABC

2019-07-22

翻了一下SQLAlchemy的官方文档,看到一个简单的办法,作业如下:

'''

作业:...

展开 ▾



2



ABC

2019-07-22

文章中的示例代码.完整可运行.

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

展开 ∨



👍 2



**夜路破晓**

2019-07-22

框架对实体的映射不难理解,数据库本身就是对现实世界的映射,借由映射将事实转换为数据.代码部分有些基础的也不难理解;基础较弱硬钢的亲们,耐心一条条来缕也可以捋顺,都是基础的东西,无非花费时间长短问题.有几个坑这里记录下,供后来人借鉴:

1.关于初始化连接数据库问题.create\_engine的参数这块容易卡壳,可以参考以下文字说明:  
create\_engine("数据库类型+数据库驱动://数据库用户名:数据库密码@IP地址:端口/数...

展开 ∨



👍 2



**Destroy、**

2019-07-22

```
rows = session.query(Player).filter(Player.height==2.08).all()
for row in rows:
    row.height = 2.09
session.commit()
session.close()
```

展开 ∨



👍 2



**Geek\_5d805b**

2019-07-25

问一下,我们已经创建过了player表,能说一下对于已有的数据库表,怎么直接将存储模型转换为数据模型吗,而不是再按字段新建

展开 ∨



👍 1



**ABC**

2019-07-22

另外,SQLAlchemy和MyBatis有点像,唯一不同的是MyBatis可以把SQL语句写在XML文件里面(当然也可以写在Java方法上),SQLAlchemy好像只能用字符串方式(在官网暂时没找到其它方式的示例)来写SQL语句.



1



许童童

2019-07-22

老师你好,能否多讲一些ORM框架的缺点,以及为什么互联网项目大多不用ORM的原因?



1



阿锋

2019-07-22

上面那个分组查询,按照分组后数据行数递增的顺序进行排序,怎么结果是[(1001, 20), (1002, 17)],那不是递减?是不是写错了?

展开

作者回复: rows = session.query(Player.team\_id, func.count(Player.player\_id)).group\_by(Player.team\_id).having(func.count(Player.player\_id)>5)

这里使用的是asc(),所以结果应该是: [(1002, 17), (1001, 20)],你可以再check下order\_by的部分



1



大牛凯

2019-07-22

老师好,对于修改数据的事例有一点困惑还请您解答。对于下面这段代码中

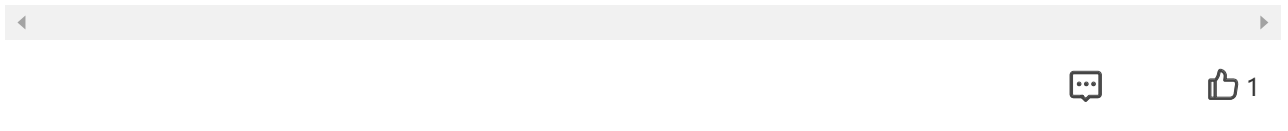
```
row = session.query(Player).filter(Player.player_name=='索恩 - 马克').first()
row.height = 2.17
session.commit()...
```

展开

作者回复: 我们对数据表进行的增删改查实际上都是通过 session对象来完成的。这里row是定位 session对象中想要查询的位置,然后对row.height进行了修改,也就是对session对象中那个查



询位置的height进行了修改。这时数据表中的内容还没有更新，需要采用session.commit()来完成持久化。所以重点是我们修改了session对象的数据，然后进行了session.commit()



**jxs1211**

2019-07-25

老师，我的表中有个时间字段，我想在插入数据时，自动生成时间应怎么设置该字段，是这样吗：

```
`create_time` timestamp(0) NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
```



**Geek\_5d805b**

2019-07-25

to\_dict方法这块看不太懂，base类指的是player类吗，谁给讲讲



**Nixus**

2019-07-24

ORM的使用，更多的不都是通过查文档的吗？

展开 ∨

