**ORM关系之一对多：**

mysql级别的外键，还不够爽，必须拿到一个表的外键，然后通过这个外键再去另外一张表中查找，这样太麻烦了。

SQLAlchemy提供了一个`relationship`，这个类可以定义属性，以后在访问相关联的表的时候就直接可以通过属性访问的方式就可以访问得到了。另外，可以通过`backref`来指定反向访问的属性名称。newss是指有多篇新闻。他们之间的关系是一个“一对多”的关系。

**演示代码如下：**

|  |
| --- |
| **from** sqlalchemy **import** create\_engine,Column,Integer,String,Float,Enum,Boolean,DECIMAL,Text,\  Date,DateTime,Time,func,and\_,or\_,ForeignKey  **from** sqlalchemy.ext.declarative **import** declarative\_base  **from** sqlalchemy.dialects.mysql **import** LONGTEXT  **from** sqlalchemy.orm **import** sessionmaker,relationship  **import** random  **import** enum  **from** datetime **import** date  **from** datetime **import** datetime  **from** datetime **import** time  #准备数据库的一堆信息 ip port user pwd 数据库的名称 按要求组织格式  HOSTNAME = **'127.0.0.1'**  PORT = **'3306'**  DATABASE = **'first\_sqlalchemy'**  USERNAME = **'root'**  PASSWORD = **'root'**  #dialect+driver://username:password@host:port/database?charset=utf8  #按照上述的格式来 组织数据库信息  DB\_URI =**"mysql+pymysql://{username}:{password}@{host}:{port}/{db}?charset=utf8"**.\  format(username=USERNAME,password=PASSWORD,host=HOSTNAME,port=PORT,db=DATABASE)  #创建数据库引擎  engine = create\_engine(DB\_URI)  #创建会话对象  session = sessionmaker(engine)()  #创建ORM模型  Base = declarative\_base(engine)  # 主表 / 从表  # user/news  **class** User(Base):  \_\_tablename\_\_ = **'user'**  id = Column(Integer,primary\_key=**True**,autoincrement=**True**)  uname = Column(String(50),nullable=**False**)  # newss=relationship("News") #这种写法不是最优的，通常会把它通过反向声明的方式写在“多”的那一方  **def** \_\_repr\_\_(self):  **return "<User(uname:%s)>"** % self.uname  **class** News(Base):  \_\_tablename\_\_ = **'news'**  id = Column(Integer,primary\_key=**True**,autoincrement=**True**)  title = Column(String(50),nullable=**False**)  content = Column(Text,nullable=**False**)  #外键  uid = Column(Integer,ForeignKey(**"user.id"**))  #正向author = relationship("User")  #正向 和 反向在一起 表明两个模型之间的关系  author = relationship(**"User"**,backref=**"newss"**)  **def** \_\_repr\_\_(self):  **return "<News(title:%s,content=%s)>"** % (self.title,self.content)  # Base.metadata.drop\_all()  # Base.metadata.create\_all()  #需求1：查询 第一篇新闻的 作者是谁  # news= session.query(News).first()  # print(news)  # print(news.uid) #1  # user = session.query(User).get(news.uid)  # print(user.uname)  #上述的需求 能够被实现 但是太麻烦，引入relationship进行查询优化  # news = session.query(News).first()  # print(news.author)  # print(news.author.uname)  #需求2：查询xx作者的所有文章  user = session.query(User).first()  print(user.newss) |