**join的使用\_高级查询之多表查询：**

**1.mysql中的外连接、内连接**

表A数据：

aID         aNum

1           a20050111

2           a20050112

3           a20050113

4           a20050114

5           a20050115

表B数据:

bID         bName

1           2006032401

2           2006032402

3           2006032403

4           2006032404

8           2006032408

**1.1 left join(左连接)**

SELECT \* FROM a

LEFT JOIN  b

ON a.aID =b.bID

结果如下:

aID         aNum           bID            bName

1            a20050111         1              2006032401

2            a20050112         2              2006032402

3            a20050113         3              2006032403

4            a20050114         4              2006032404

5            a20050115         NULL        NULL

结果说明:

  left join是以A表的记录为基础的,A可以看成左表,B可以看成右表,leftjoin是以左表为准的.

换句话说,左表(A)的记录将会全部表示出来,而右表(B)只会显示符合搜索条件的记录(例子中为: A.aID = B.bID).

B表记录不足的地方均为NULL.

**1.2 right join(右连接)**

SELECT  \* FROM a

RIGHT JOING b

ON a.aID = b.bID

结果如下:

aID         aNum          bID            bName

1            a20050111         1              2006032401

2            a20050112         2              2006032402

3            a20050113         3              2006032403

4            a20050114         4              2006032404

NULL     NULL           8              2006032408

结果说明:

  仔细观察一下,就会发现,和left join的结果刚好相反,这次是以右表(B)为基础的,A表不足的地方用NULL填充.

**1.3 inner join(相等连接或内连接)**

SELECT \* FROM  a

INNER JOIN  b

ON a.aID =b.bID

等同于以下SQL句:

SELECT \*

FROM a,b

WHERE a.aID = b.bID

结果如下:

aID         aNum          bID            bName

1            a20050111         1              2006032401

2            a20050112         2              2006032402

3            a20050113         3              2006032403

4            a20050114         4              2006032404

结果说明:

  很明显,这里只显示出了 A.aID = B.bID的记录.这说明inner join并不以谁为基础,它只显示符合条件的记录.

**2.join的使用\_高级查询：**

1. join分为left join（左外连接）和right join（右外连接）以及内连接（等值连接）。

2. 在sqlalchemy中，使用join来完成内连接。在写join的时候，如果不写join的条件，那么默认将使用外键来作为条件连接。

3. 查询出来的字段，跟join后面的东西无关，而是取决于query方法中传了什么参数。（模型名=全表；模型名.属性=表名.字段）。

4. 在sqlalchemy中，使用outerjoin来完成外连接（默认是左外连接）。

**3.代码演示**

|  |
| --- |
| **from** sqlalchemy **import** create\_engine,Column,Integer,Float,Boolean,DECIMAL,Enum,\  Date,DateTime,Time,String,Text,func,or\_,and\_,ForeignKey,Table  **from** sqlalchemy.dialects.mysql **import** LONGTEXT  **from** sqlalchemy.ext.declarative **import** declarative\_base  **from** sqlalchemy.orm **import** sessionmaker,relationship,backref  **import** random,time  **from** datetime **import** datetime  HOSTNAME = **'127.0.0.1'**  PORT = **'3306'**  DATABASE = **'first\_sqlalchemy'**  USERNAME = **'root'**  PASSWORD = **'root'**  DB\_URI =**"mysql+pymysql://{username}:{password}@{host}:{port}/{db}?charset=utf8"**.format(username=USERNAME,password=PASSWORD,host=HOSTNAME,port=PORT,db=DATABASE)  engine = create\_engine(DB\_URI)  Base = declarative\_base(engine)  session = sessionmaker(engine)()  **class** User(Base):  \_\_tablename\_\_ = **'user'**  id = Column(Integer,primary\_key=**True**,autoincrement=**True**)  uname = Column(String(50),nullable=**False**)  **def** \_\_repr\_\_(self):  **return "<User(uname: %s)>"** % self.uname  **class** Article(Base):  \_\_tablename\_\_ = **'article'**  id = Column(Integer, primary\_key=**True**, autoincrement=**True**)  title = Column(String(50), nullable=**False**)  create\_time = Column(DateTime, nullable=**False**, default=datetime.now)  uid = Column(Integer,ForeignKey(**"user.id"**))  author = relationship(**"User"**,backref=**"articles"**)  **def** \_\_repr\_\_(self):  **return "<Article(title: %s)>"** % self.title  **def** add\_data():  Base.metadata.drop\_all()  Base.metadata.create\_all()  user1 = User(uname=**'莫莫'**)  user2 = User(uname=**'露露'**)  **for** x **in** range(1):  article = Article(title=**'title %s'** % x)  article.author = user1  session.add(article)  session.commit()  **for** x **in** range(1,3):  article = Article(title=**'title %s'** % x)  article.author = user2  session.add(article)  session.commit()  #找到所有的用户，按照发表的文章数量进行排序  #注意1：在sqlalchemy中，使用join来完成内连接  **def** oper1():  #select user.uname,count(article.id) from user join article on user.id=article.uid group by user.id order by count(article.id) desc;  # result = session.query(User.uname, func.count(Article.id)).join(Article,User.id==Article.uid).group\_by(User.id).order\_by(  # func.count(Article.id).desc())  # print(result) #sql语句  result = session.query(User.uname, func.count(Article.id)).join(Article, User.id == Article.uid).group\_by(  User.id).order\_by(  func.count(Article.id).desc()).all()  print(result) # 结果:列表  #找到所有的用户，按照发表的文章数量进行排序  #注意2：在写join的时候，如果不写join的条件，那么默认将使用外键来作为条件连接。  **def** oper2():  result = session.query(User.uname, func.count(Article.id)).join(Article).group\_by(  User.id).order\_by(  func.count(Article.id).desc()).all()  print(result) # 结果:列表  #找到所有的用户，按照发表的文章数量进行排序  #注意3： 查询出来的字段，跟join后面的东西无关，而是取决于query方法中传了什么参数。（模型名=全表；模型名.属性=表名.字段）  **def** oper3():  result = session.query(User).join(Article).group\_by(  User.id).order\_by(  func.count(Article.id).desc()).all()  print(result) # 结果:列表  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  # add\_data()  # oper1()  # oper2()  oper3() |