数据库操作

BBCLOUD 林凡

https://github.com/lvancer/course_python

lin029011@163.com

LV5

大纲

数据库

SQL

操作

ORM

练习

lin029011@163.com

LV5

数据库

数据库包含关系型数据库和非关系型数据库。

关系型：SQLite、MySQL、Oracle、SqlServer等。

非关系型：Hbase、MonogoDB、Elasticsearch、redis等。

我们今天介绍的是关系型数据库的操作。

关系型数据库

数据库：包含多张数据表。

数据表：包含行【记录】和列【字段】。

列字段：代表用于描述记录的一个属性。

行记录：用多个字段进行描述的一条数据。

都使用SQL语句进行查询。

lin029011@163.com

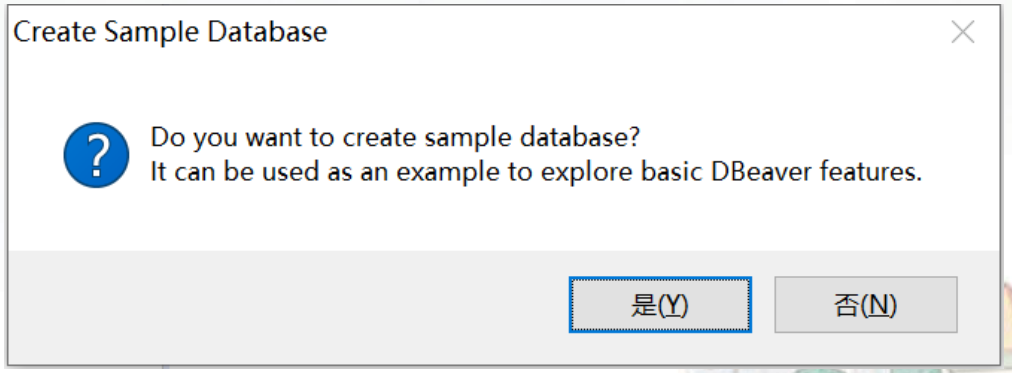
数据库

数据库工具：DBeaver，一个免费的通用数据库工具。

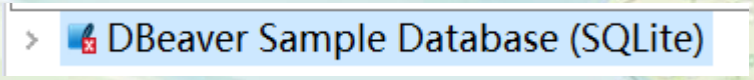
下载地址：https://dbeaver.io/download/

安装完成后，打开后先不打开任何数据库。

看到下面对话框后，点击是。



然后工具就会自动帮我们创建好一个作为示例的SQLite数据库。



lin029011@163.com

LV5

数据库

安装驱动：双击数据库，下载安装驱动。

有时下载速度很慢或者下载失败，

需要更换国内源。

lin029011@163.com

LV5

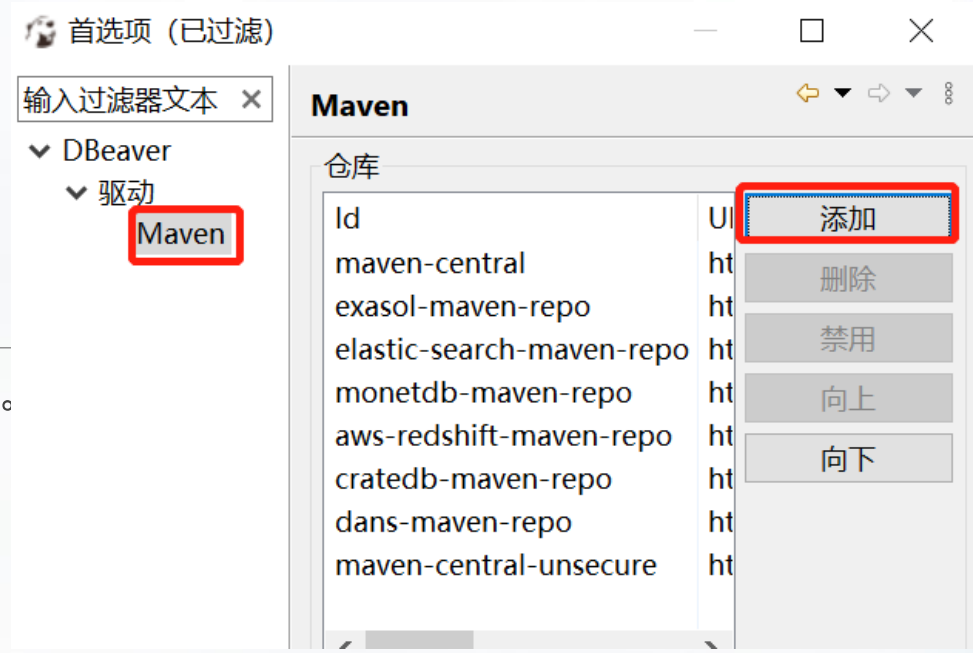
数据库

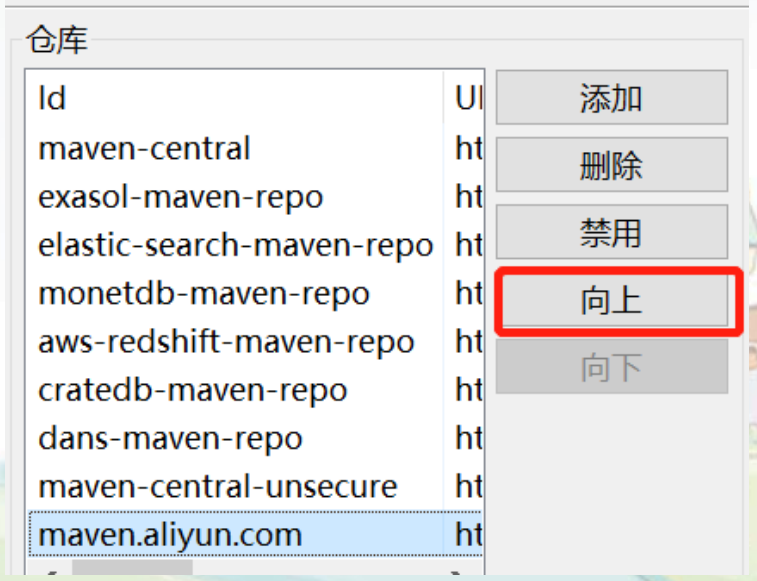
DBeaver是Java开发的，驱动使用的是Java的包管理maven。

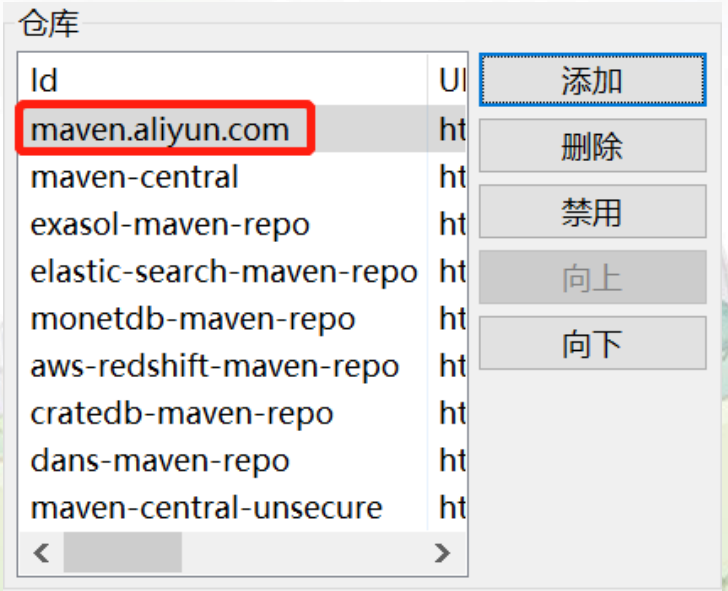
如果速度慢，可以点击下载配置。选择Maven => 添加。

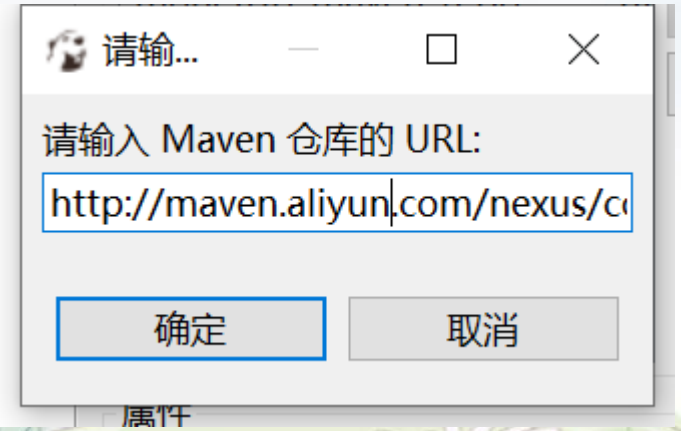
输入http://maven.aliyun.com/nexus/content/groups/public/

接着选择刚刚添加的源，点击向上，直到顶部，最后重启DBeaver。









lin029011@163.com

数据库

创建数据库连接：

数据库 => 新建连接 => 选择数据库类型

创建好后，一样安装驱动就可以使用了。

lin029011@163.com

LV5

SQL

SQL语句是操作数据库的语言。这里只做简单介绍。

包括INSERT【插入】、SELECT【查询】、UPDATE【更新】、DELETE【删除】。

INSERT

```
INSERT INTO foo (bar, baz) VALUES (1, 'a');
```

INSERT INTO 表名 (字段1, 字段2, ….) VAULES (值1, 值2, …);

lin029011@163.com

SQL

UPDATE：

```
UPDATE foo SET baz = 'bb' WHERE bar = 1;
```

UPDATE 表名 SET 字段 = 新的值 WHERE 条件字段1 = 值1, 条件字段2 = 值2 … ;

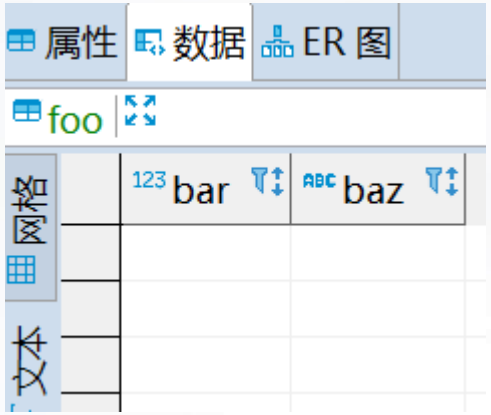
lin029011@163.com

LV5

SQL

DELETE：

```
DELETE FROM foo WHERE bar = 1;
```



DELETE FROM 表名 WHERE 条件字段1 = 值1, 条件字段2 = 值2… ;

lin029011@163.com

LV5

SQL

SELECT：

查询所有

```
SELECT * FROM Artist;
```

条件查询

```
SELECT * FROM Track WHERE AlbumId = 1;
```

部分字段

```
SELECT TrackId, Name FROM Track WHERE AlbumId = 1;
```

查询排序

```
SELECT * FROM Track WHERE AlbumId = 1 ORDER BY Bytes;
```

```
SELECT * FROM Track WHERE AlbumId = 1 ORDER BY Bytes DESC;
```

SELECT */字段 FROM 表名 WHERE 条件 ORDER BY 排序字段；

lin029011@163.com

LV5

SQL

关系型：

每个表都有一个主键，来唯一标识这条记录，

如Track表中的TrackId和Album表中的AlbumId。

外键则是表示表与表之间关系。

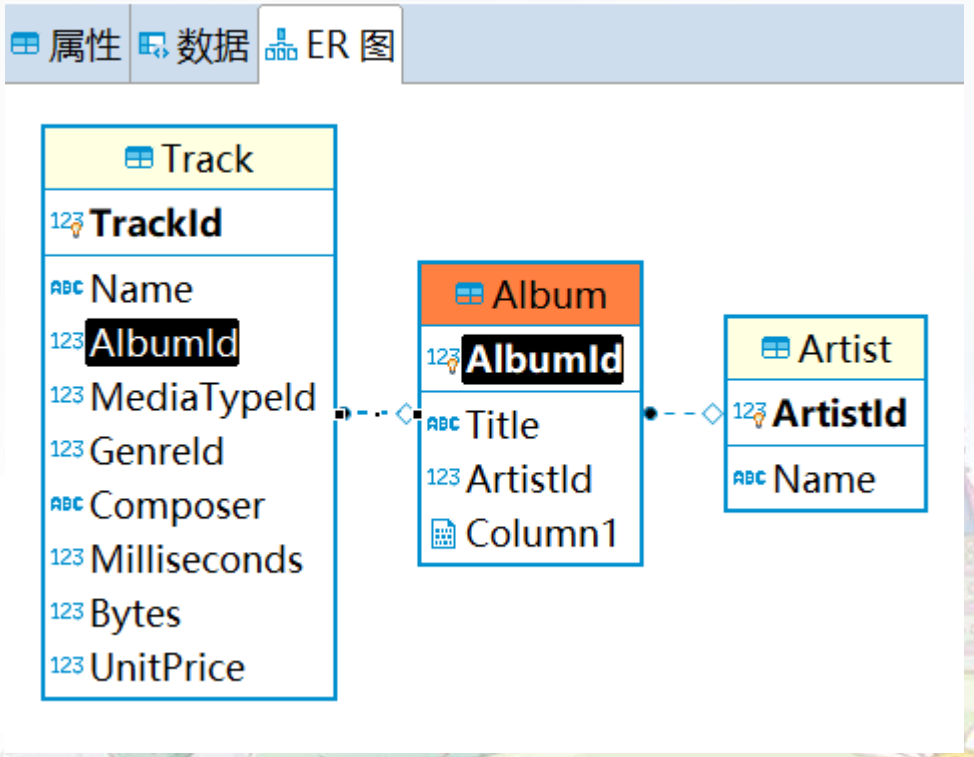
Track表中的AlbumId就是外键，

它关联的是Album这张表中的一条数据。

Album表中的ArtistId则是关联着Artist表。

主键和外键一般都是数字，

这样可以有效减少存储空间，以及方便数据的修改。



lin029011@163.com

LV5

SQL

JOIN：将有关系的表进行联合查询。

```
SELECT t.Name, a.Title FROM Track t JOIN Album a WHERE t.AlbumId = a.AlbumId;
```

Track(+)

SELECT t.Name, a.Title FROM Tra 输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)

网格 文本

	Name	Title
1	For Those About To Rock (We Salute You)	For Those About To Rock We Salute You
2	Balls to the Wall	Balls to the Wall
3	Fast As a Shark	Restless and Wild
4	Restless and Wild	Restless and Wild
5	Princess of the Dawn	Restless and Wild
6	Put The Finger On You	For Those About To Rock We Salute You
7	Let's Get It Up	For Those About To Rock We Salute You
8	Inject The Venom	For Those About To Rock We Salute You
9	Snowballed	For Those About To Rock We Salute You
10	Evil Walks	For Those About To Rock We Salute You
11	C.O.D.	For Those About To Rock We Salute You
12	Breaking The Rules	For Those About To Rock We Salute You

JOIN前后联合两张表，t和a是别名方便使用。

lin029011@163.com

LV5

SQL

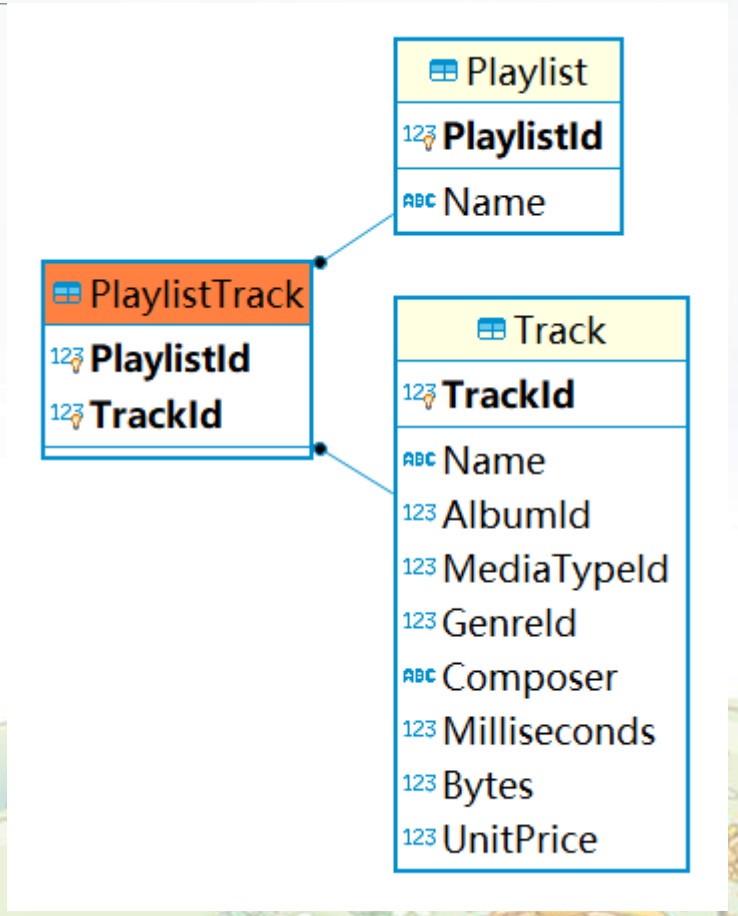
多对多关系：

一张包含两个外键的表，使两张表形成多对多关系。

即一个Playlist里有多个Track，一个Track也在多个Playlist里。

```
SELECT p.Name Playlist, t.Name Track FROM Playlist p
JOIN PlaylistTrack pt JOIN Track t ON pt.PlaylistId =
p.PlaylistId AND pt.TrackId = t.TrackId;
```

	Playlist	Track
1	Music	Band Members Discuss Tracks from "Revelations"
2	Music	Revelations
3	Music	One and the Same
4	Music	Sound of a Gun
5	Music	Until We Fall



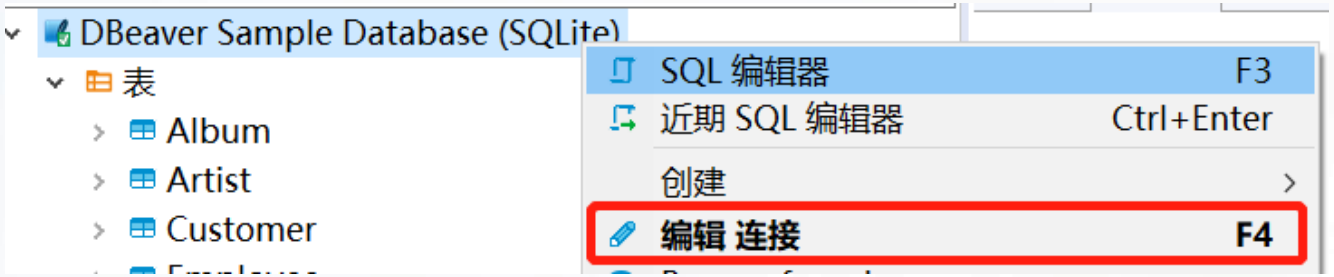
lin029011@163.com

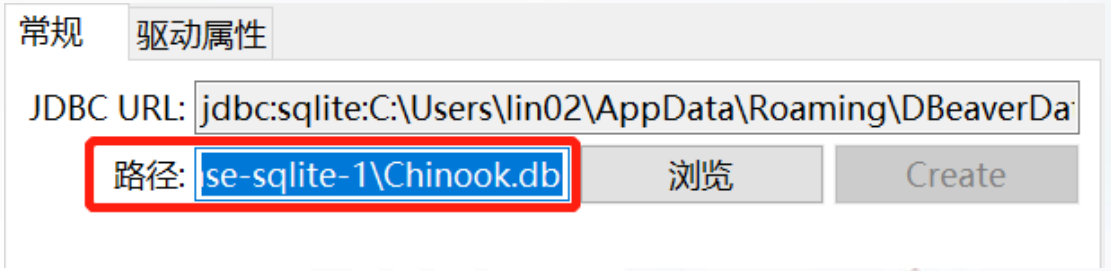
LV5

操作

SQLite：内置的sqlite3库。

SQLite数据库就是一个文件，我们在右键编辑连接中找到路径。【记得修改 \ 为 \\】





创建连接

```
# 数据库地址
dbfile = "xxx.db"
conn = sqlite3.connect(dbfile)          # 连接数据库
```

lin029011@163.com

LV5

操作

增删改：

```
cursor = conn.cursor()                          # 数据库指针，用于操作
# 执行语句
cursor.execute("INSERT INTO foo (bar, baz) VALUES (1, 'a');")
if cursor.rowcount > 0:      # 有增加删除修改的行数
    print('修改了' + str(cursor.rowcount) + '行')
conn.commit()                # 执行
cursor.close()           # 关闭指针
```

每次进行操作前都要创建一个指针cursor。

之后执行指针的execute方法，里面直接写入SQL语句。可以多执行几个SQL语句。

rowcount用于判断是否有改变表的内容，大于0就是有修改。

最后conn.commit方法执行语句。使用完指针，将指针关闭cursor.close。

lin029011@163.com

操作

关闭连接：当不需要数据库后，要将数据库关闭。

```
conn.close()                # 关闭连接
```

查询：

```
cursor = conn.cursor()
cursor.execute("SELECT * FROM foo;")
result = cursor.fetchall()
print(result)
cursor.close()              # 关闭指针
```

通过fetchall方法获取查询的结果，结果使用列表返回。 `[(1, 'a')]`

最后同样要关闭指针和连接。

lin029011@163.com

LV5

操作

参数格式化：执行SQL时，也可以进行类似格式化的方式编写。

```
cursor.execute("SELECT * FROM Track WHERE AlbumId = ? AND TrackId > ?;", (1, 1)
```

使用问号代替条件的值，第二个参数写入一个元组，包含具体的值用于依次替换问号。

MySQL：pymysql模块。

除了使用不同的模块，

创建出对应的连接后，

剩余的操作都是一样的。

```
import pymysql
conn = pymysql.connect(
    host='127.0.0.1',          # 数据库地址
    port=3306,                 # 数据库端口
    user='root',               # 用户名
    password='123456',         # 密码
    db='test1',                # 数据库名
)
```

lin029011@163.com

LV5

ORM

ORM是一种面向对象的方式进行数据库的操作。

近来，ORM已经是一种很常用的方式，因为比较简单，不需要写任何的SQL语句。

Python中主要的ORM框架是SQLAlchemy和Django中的ORM。

这里介绍SQLAlchemy，Django的ORM在Django课程中再介绍。

SQLAlchemy：sqlalchemy模块。

创建连接，使用create_engine方法创建对应的引擎。

```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///file.db')                    # sqlite3
engine = create_engine('mysql+pymsyql://root:123456@127.0.0.1/test') # mysql
```

参数使用了数据库连接字符串。

lin029011@163.com

LV5

ORM

创建表的基类：这是数据库里所有表的父类。

```
from sqlalchemy.ext.declarative import declarative_base
Base = declarative_base()
```

创建表：

```
from sqlalchemy import Column, String, Integer  # 引入字段定义类
class Foo(Base):                                # 继承基类

  __tablename__ = 'foo'                         # 定义表名

  bar = Column(Integer, primary_key=True) # 定义字段，数字，主键必须定义
  baz = Column(String(20))                      # 20长度的字符串
```

Column定义字段，参数1定义字段类型，primary_key定义主键。【必须有一个主键】

lin029011@163.com

LV5

ORM

添加：

任何操作前，要先创建好一个Session类。

```
Session = sessionmaker(bind=engine)      # 创建Session类
```

```
session = Session()                      # 创建session
foo = Foo(bar=1, baz='a')                # 新增数据
session.add(foo)                         # 添加
session.commit()                         # 执行
session.close()                          # 关闭session
```

这样的新增数据就完全不需要写入SQL，而直接进行对象操作即可。

新建一个Foo对象就是一条要添加的数据，之后进行添加执行。

Foo的参数是我们之前定义好的字段。

lin029011@163.com

LV5

ORM

修改：

```
session = Session()
foo = session.query(Foo).filter_by(bar=1).first()  # 条件查询
foo.baz = 'bb'        # 直接修改属性
session.add(foo)      # add保存
session.commit()
session.close()
```

修改前要先用query查询Foo表，filter_by里是条件语句，最后的first表示获得一条。

另一个all方法获得所有满足条件的数据，返回一个列表。

查询到后，直接进行属性修改，之后保存即可。

效率上比直接的SQL要慢，但是开发简单，在大部分场合可以这样用。

lin029011@163.com

LV5

ORM

删除：

```
session = Session()
foo = session.query(Foo).filter_by(bar=1).delete()
session.commit()
session.close()
```

与修改一样，我们像先查询，这里使用delete方法删除满足条件数据。

lin029011@163.com

LV5

ORM

查询：先定义好几张表。

```
class Artist(Base):

    __tablename__ = 'Artist'

    ArtistId = Column(Integer, primary_key=True, nullable=False)
    Name = Column(String(120))
```

```
class Album(Base):

    __tablename__ = 'Album'

    AlbumId = Column(Integer, primary_key=True, nullable=False)
    Title = Column(String(160), nullable=False)
    ArtistId = Column(Integer, nullable=False)
```

163.com

LV5

ORM

```
class Track(Base):

    __tablename__ = 'Track'

    TrackId = Column(Integer, primary_key=True, nullable=False)
    Name = Column(String(200), nullable=False)
    AlbumId = Column(Integer)
    Bytes = Column(Integer)
```

```
session = Session()
artists = session.query(Artist).all()      # 查询全部
tracks = session.query(Track).filter_by(AlbumId=1).all()    # 条件查询
tracks = session.query(Track)\
    .with_entities(Track.TrackId, Track.Name)\
    .filter_by(AlbumId=1).all()    # 部分字段
tracks = session.query(Track)\
    .order_by(Track.Bytes)\
    .filter_by(AlbumId=1).all()    # 排序
session.close()
```

filter_by定义条件语句，with_entities指定返回字段，order_by排序。

lin029011@163.com

LV5

ORM

外键：ForeignKey。

```
from sqlalchemy import ForeignKey
from sqlalchemy.orm import relationship
class Album(Base):
    # 外键
    ArtistId = Column(Integer, ForeignKey('Artist.ArtistId'), nullable=False)
    Artist = relationship("Artist", backref="Album")
class Track(Base):
    # 外键
    AlbumId = Column(Integer, ForeignKey('Album.AlbumId'))
    Album = relationship("Album", backref="Track")
```

Column的第二个参数定义外键ForeignKey类，其参数为关联的表名.字段。

然后再定义个外键关联的实体，relationship(关联表，backreef=本表)。

lin029011@163.com

LV5

ORM

查询时，就可以直接获得外键的实体。

不用join就可以获得外键的信息。

```
session = Session()
album = session.query(Album).first()
artist = album.Artist   # 外键实体
print(artist.Name)
session.close()
```

```
SELECT t.Name, a.Title FROM Track t JOIN Album a WHERE t.AlbumId = a.AlbumId;
```

```
session = Session()
tracks = session.query(Track).all()
ret = []
for track in tracks:
    ret.append((track.Name, track.Album.Title))
print(ret)
session.close()
```

lin029011@163.com

LV5

ORM

多对多：

先添加Playlist表，定义Tracks字段。

```
class Playlist(Base):

    __tablename__ = 'Playlist'

    PlaylistId = Column(Integer, primary_key=True, nullable=False)
    Name = Column(String(200), nullable=False)
    # 多对多
    Tracks = relationship("Track", backref="Playlist", secondary=playlist_trac
```

与外键唯一的不同是多了一个secondary参数，playlist_track是我们要定义的中间关联表。

lin029011@163.com

LV5

ORM

中间关联表：

```
from sqlalchemy import Table
playlist_track = Table(
    "PlaylistTrack",          # 表名
    Base.metadata,            # 表类型
    Column("PlaylistId", Integer,
           ForeignKey("Playlist.PlaylistId"), primary_key=True),
    Column("TrackId", Integer,
           ForeignKey("Track.TrackId"), primary_key=True)
)
```

使用Table直接定义表，定义好两个外键，且作为联合主键。

lin029011@163.com

LV5

ORM

多对多查询：

```
session = Session()
playlist = session.query(Playlist).first()
print(playlist.Name)
for track in playlist.Tracks:
    print('\t{} 【专辑:{}】 【作者: {}】'
          .format(track.Name, track.Album.Title, track.Album.Artist.Name))
session.close()
```

通过这个方式，我们不仅可以很方便获得两张表的关联，

还可以很方便的获得更多的关联，而不去需要进行复杂的JOIN操作。

lin029011@163.com

LV5

ORM

	PlaylistId	TrackId
1	1	3,503
2	5	3,503
3	8	3,503
4	12	3,503
5	13	3,503
6	2	3,503

多对多增加：

```
session = Session()
playlist = session.query(Playlist).filter_by(PlaylistId=2).first()
track = session.query(Track).filter_by(TrackId=3503).first()
playlist.Tracks.append(track)
session.add(playlist)
session.commit()
session.close()
```

查询获得要添加的playlist和被添加track。

直接通过playlist.Tracks.append添加，之后保存执行即可。

lin029011@163.com

LV5

练习

1、自己进行SQL的练习，并用Python实现。

2、尝试将Sample数据库的ORM表全部补齐，

并查询出1个Artist所有Album的所有Track的所有购买信息。

lin029011@163.com