

# Python开发

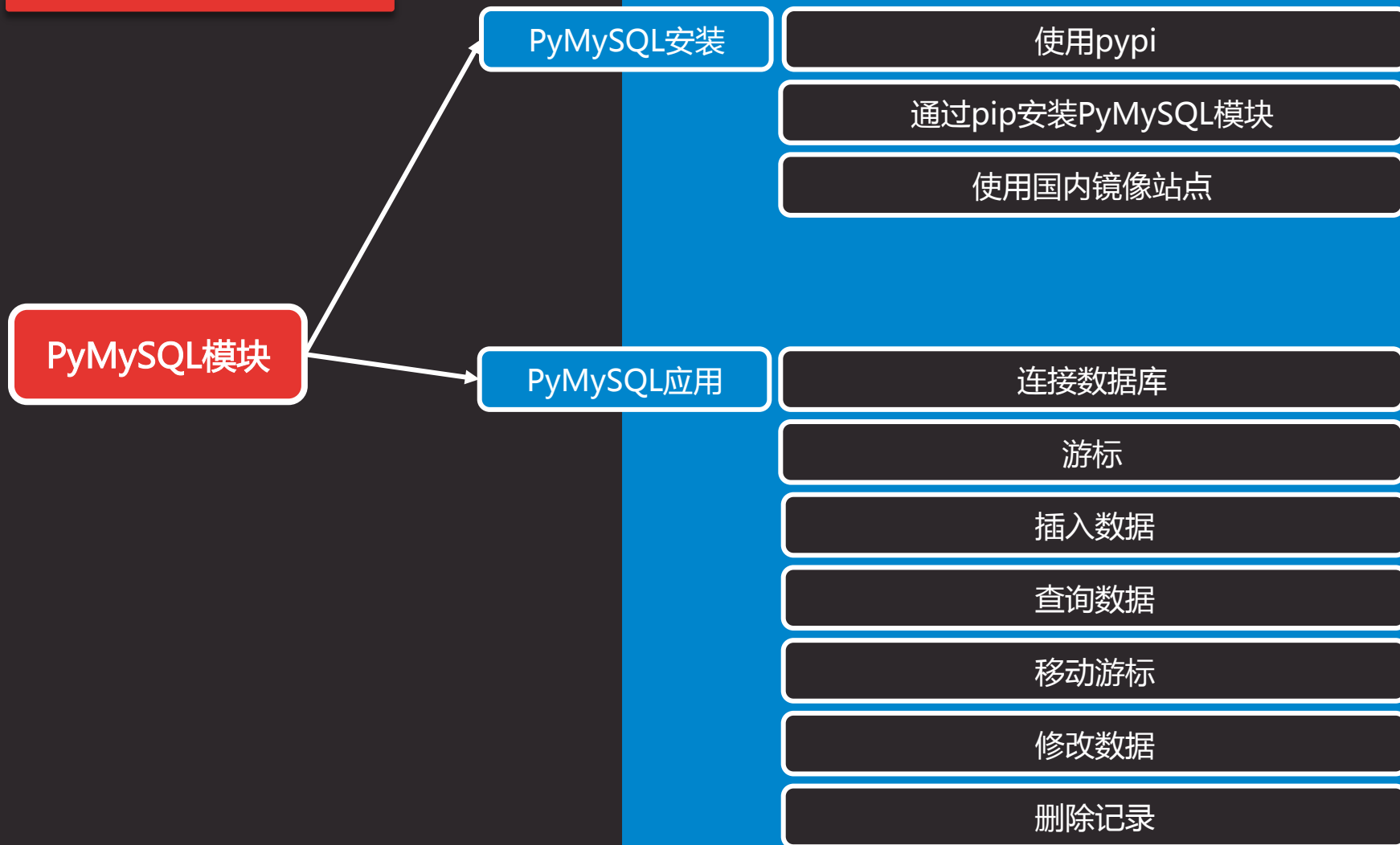
NSD PYTHON

# 内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	PyMySQL模块
	10:30 ~ 11:20	
	11:30 ~ 12:00	SQLAlchemy基础
下午	14:00 ~ 14:50	
	15:00 ~ 15:50	SQLAlchemy进阶
	16:10 ~ 17:00	
	17:10 ~ 18:00	总结和答疑



# PyMySQL模块



# PyMySQL安装



# 使用pypi

- pypi即python package index
- 是python语言的软件仓库
- 官方站点为<https://pypi.python.org>



# 通过pip安装PyMySQL模块

- 安装依赖包

```
[root@localhost packages]# yum install -y gcc
```

- 本地安装

```
[root@localhost packages]# pip3 install PyMySQL-0.8.0.tar.gz
```

- 在线安装

```
[root@localhost packages]# pip3 install pymysql
```



# 使用国内镜像站点

- 为了实现安装加速，可以配置pip安装时采用国内镜像站点

```
[root@localhost ~]# mkdir ~/.pip/  
[root@localhost ~]# vim ~/.pip/pip.conf  
[global]  
index-url=http://pypi.douban.com/simple/  
[install]  
trusted-host=pypi.douban.com
```



# PyMySQL应用





# 连接数据库

- 创建连接是访问数据库的第一步

```
conn = pymysql.connect(host='127.0.0.1', port=3306, user='root',  
passwd='tedu.cn', db='tkq1', charset='utf8')
```



# 游标

- 游标 ( cursor ) 就是游动的标识
- 通俗的说，一条sql取出对应n条结果资源的接口/句柄，就是游标，沿着游标可以一次取出一行

```
cursor = conn.cursor()
```



# 插入数据

- 对数据库表做修改操作，必须要commit

```
sql1 = "insert into departments(dep_name) values(%s)"
result = cur.execute(sql1, ('development',))
```

```
sql2 = "insert into departments(dep_name) values(%s)"
data = [('hr',), ('op',)]
result = cur.executemany(sql2, data)
```

```
sql3 = "insert into departments(dep_name) values(%s)"
data = [('行政',), ('财务',), ('运营',)]
result = cur.executemany(sql3, data)
```

```
conn.commit()
```



# 查询数据

- 可以取出表中一条、多条或全部记录

```
sql4 = "select * from departments"  
cur.execute(sql4)  
result = cur.fetchone()  
print(result)
```

```
result2 = cur.fetchmany(2)  
print(result2)
```

```
result3 = cur.fetchall()  
print(result3)
```



# 移动游标

- 如果希望不是从头取数据，可以先移动游标

```
cur.scroll(1, mode="relative")  
cur.scroll(2, mode="absolute")
```

```
sql5 = "select * from departments"  
cur.execute(sql5)  
cur.scroll(3, mode='absolute')  
result4 = cur.fetchmany(2)  
print(result4)
```



# 修改数据

- 通过update修改某一字段的值

```
sql6 = "update departments set dep_name=%s where dep_name=%s"  
result = cur.execute(sql6, ('operations', 'op'))  
print(result)  
conn.commit()
```



# 删除记录

- 通过delete删除记录

```
sql7 = "delete from departments where dep_id=%s"  
result = cur.execute(sql7, (6,))  
print(result)  
conn.commit()
```



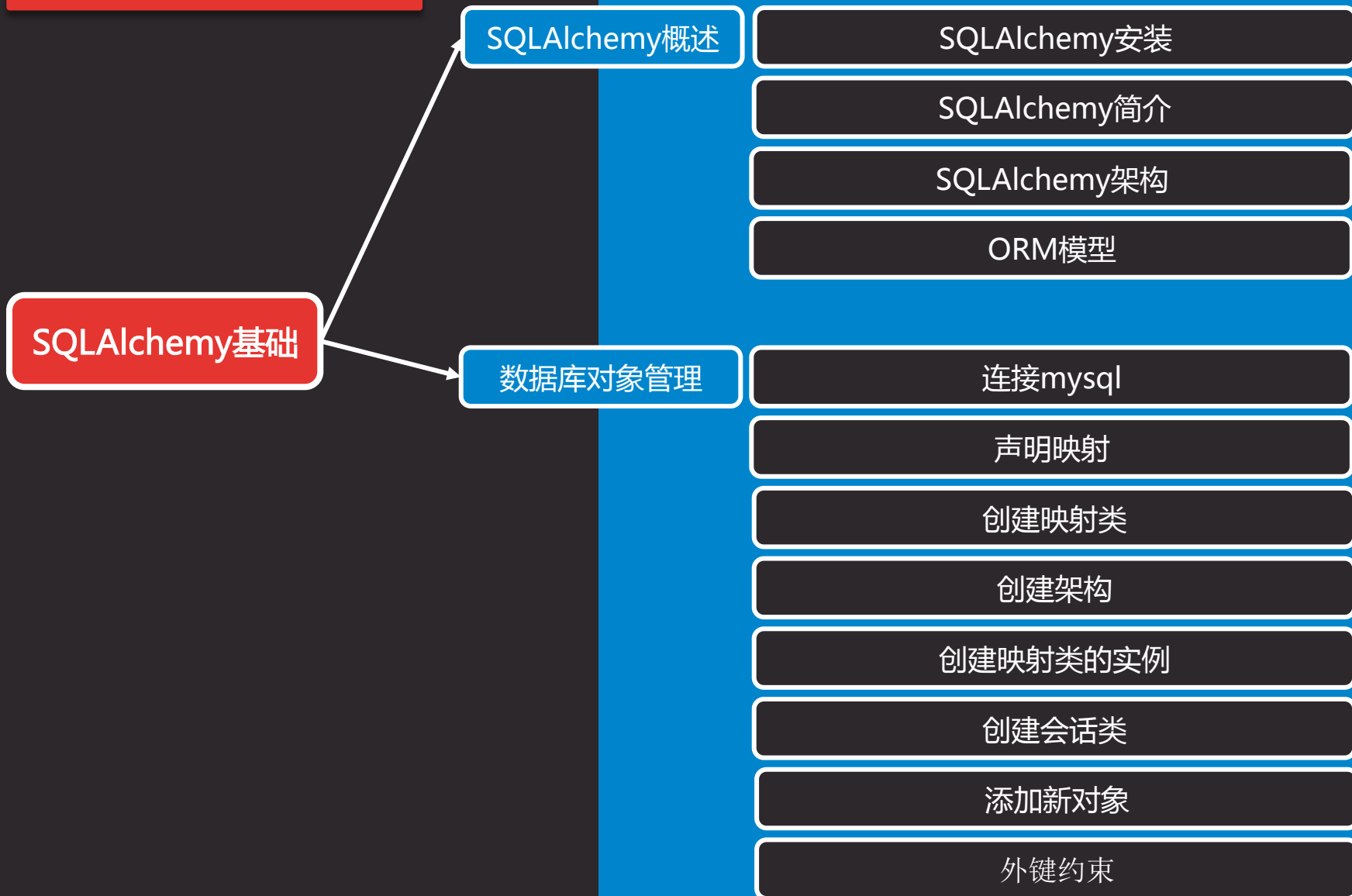
# 案例1：向表中添加数据

1. 向employees表插入数据
2. 向salary表插入数据
3. 插入的数据需要commit到数据库中





# SQLAlchemy基础



# SQLAlchemy概述

---

# 连接mysql

- 通过create\_engine实现数据库的连接

```
[root@bogon bin]# mysql -uroot -ptedu.cn
```

```
MariaDB [(none)]> create database tarena default char set utf8;
```

```
>>> from sqlalchemy import create_engine
```

```
>>> engine = create_engine('mysql+pymysql://root:tedu.cn@localhost/  
tarena', encoding='utf8', echo=True)
```

//echo=True表示将日志输出到终端屏幕，默认为False



# 声明映射

- 当使用ORM的时候，配置过程从描述数据库表开始
- 通过自定义类映射相应的表
- 通过声明系统实现类映射
- 首先通过声明系统，定义基类

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



# 创建映射类

- 一旦创建了基类，就可以创建自定义映射类了

```
>>> from sqlalchemy import Column, Integer, String
>>> class Departments(Base):
...     __tablename__ = 'departments'
...     dep_id = Column(Integer, primary_key=True)
...     dep_name = Column(String(20))
...     def __repr__(self):
...         return "<Department(dep_name='%s')>" % self.dep_name
```

//\_\_repr\_\_是可选项



# 创建架构

- 类构建完成后，表的信息将被写入到表的元数据（ metadata ）

```
>>> Departments.__table__
Table('departments', MetaData(bind=None), Column('dep_id',
Integer(), table=<departments>, primary_key=True, nullable=False),
Column('dep_name', String(), table=<departments>), schema=None)
```



## 创建架构（续1）

- 通过表的映射类，在数据库中创建表

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



# 创建映射类的实例

- 创建实例时，并不会真正在表中添加记录

```
dep_dev = Departments(dep_name='developments')  
print(dep_dev.dep_name)  
print(str(dep_dev.dep_id))
```





# 创建会话类

- ORM访问数据库的句柄被称作Session

```
>>> from sqlalchemy.orm import sessionmaker  
>>> Session = sessionmaker(bind=engine)
```

如果在创建session前还未创建engine，操作如下

```
>>> Session = sessionmaker()  
>>> Session.configure(bind=engine)    //创建engine后执行
```



# 添加新对象

- 会话类的实例对象用于绑定到数据库
- 实例化类的对象，并不打开任何连接
- 当实例初次使用，它将从Engine维护的连接池中获得一个连接
- 当所有的事务均被commit或会话对象被关闭时，连接结束

```
>>> session = Session()
>>> session.add(dep_dev)
>>> session.commit()
>>> print(str(dep_dev.dep_id))
>>> session.close()
```



## 添加新对象（续1）

- 可以创建多个实例，批量添加记录
- 如果有中文件，注意engine的参数

```
engine = create_engine('mysql+pymysql://root:tedu.cn@localhost/  
tarena?charset=utf8', encoding='utf8', echo=True)
```

```
dep_hr = Departments(dep_name='hr')  
dep_op =Departments(dep_name='operations')  
dep_finance =Departments(dep_name='财务')  
dep_xz =Departments(dep_name='行政')  
Session = sessionmaker(engine)  
session = Session()  
session.add_all([dep_hr, dep_op, dep_finance, dep_xz])  
session.commit()  
session.close()
```



# 外键约束

- ORM映射关系也可用于表间创建外键约束

```
class Employees(Base):
    __tablename__ = 'employees'

    emp_id = Column(Integer, primary_key=True)
    name = Column(String(20))
    genda = Column(String(10))
    phone = Column(String(11))
    dep_id = Column(Integer, ForeignKey('departments.dep_id'))

    def __repr__(self):
        return "<Employees(name='%s')>" % self.name
```



## 案例2：创建表

1. 创建employees表
2. 创建部门表
3. 创建salary表
4. 表间创建恰当的关系



## 案例2：添加数据

1. 分别在部门表、员工表和工资表中加入数据
2. 通过SQLAlchemy代码实现
3. 分别练习每次加入一行数据和每次可加入多行数据



# SQLAlchemy进阶

## SQLAlchemy进阶

### 查询操作

#### 基本查询

使用ORM描述符进行查询

使用命名元组

修改显示字段名

使用别名

排序

提取部分数据

结果过滤

常用过滤操作符

查询对象返回值

聚合

多表查询

### 修改操作

更新数据

删除记录

# 查询操作





# 基本查询

- 通过作用于session的query()函数创建查询对象
- query()函数可以接收多种参数

```
from myorm import Session , Departments
```

```
session = Session()
```

```
for instance in
```

```
session.query(Departments).order_by(Departments.dep_id):
```

```
    print(instance.dep_id, instance.dep_name)
```



# 使用ORM描述符进行查询

- 使用ORM描述符进行查询
- 返回值是元组

```
from myorm import Employees, Session
```

```
session = Session()
```

```
for name, phone in session.query(Employees.name, Employees.phone):  
    print(name, phone)
```



# 使用命名元组

- 查询对象返回的是一个命名元组
- 名称是类的名字，或是类中属性的名字

```
from myorm import Session , Departments
```

```
session = Session()
```

```
for row in session.query(Departments,Departments.dep_name):  
    print(row.Departments, row.dep_name)
```



# 修改显示字段名

- 显示的字段名可以通过label()函数进行修改

```
from myorm import Session , Departments
```

```
session = Session()
```

```
for row in session.query(Departments.dep_name.label('部门')):  
    print(row.部门)
```



# 排序

- 通过order\_by()函数可以实现按指定字段排序

```
from myorm import Session , Departments
```

```
session = Session()
```

```
for instance in
```

```
session.query(Departments).order_by(Departments.dep_id):
```

```
    print(instance.dep_id, instance.dep_name)
```



# 提取部分数据

- 通过“切片”的方式，实现部分数据的提取

```
from myorm import Session , Departments
```

```
session = Session()
```

```
for row in session.query(Departments, Departments.dep_name)[2:5]:  
    print(row.Departments, row.dep_name)
```



# 结果过滤

- 通过filter()函数实现结果过滤

```
from myorm import Session , Departments
```

```
session = Session()
```

```
for row in
```

```
session.query(Departments.dep_name).filter(Departments.dep_id==2):  
    print(row.dep_name)
```



## 结果过滤（续2）

- filter()函数可以叠加使用

```
from myorm import Session , Salary
```

```
session = Session()
```

```
for row in session.query(Salary.emp_id, Salary.base, Salary.award)\  
    .filter(Salary.award>2000).filter(Salary.base>10000):  
    print(row.emp_id)
```





# 常用过滤操作符

- 相等

```
query.filter(Employees.name=='john')
```

- 不相等

```
query.filter(Employees.name!='john')
```

- 模糊查询

```
query.filter(Employees.name.like('%j'))
```



## 常用过滤操作符（续1）

- in  
`query.filter(new_emp.name.in_(['bob', 'john']))`
- not in  
`query.filter(~new_emp.name.in_(['bob', 'john']))`
- 字段为空  
`query.filter(new_emp.name.is_(None))`
- 字段不为空  
`query.filter(new_emp.name.isnot(None))`



## 常用过滤操作符（续2）

- 多重条件and

```
from sqlalchemy import and_  
query.filter(and_(new_sal.basic>=10000,  
new_sal.extra>=2000))
```

- 多重条件or

```
from sqlalchemy import or_  
query.filter(or_(new_sal.basic>=10000,  
new_sal.extra>=3000))
```



# 查询对象返回值

- all()返回列表
- first()返回结果中的第一条记录
- one()取出所有记录，如果不是一条记录则抛出异常
- scalar()调用one()，返回第一列的值



# 聚合

- 通过count()方法，统计行数

```
print(session.query(Departments).count())
```



# 多表查询

- 通过join()方法实现多表查询

```
q = session.query(
    Employees.name, Departments.dep_name).join(Departments,
    Employees.dep_id==Departments.dep_id
)
print(q.all())
```



# 更新数据

- 通过会话的update()方法更新

```
from myorm import Session, Departments
```

```
session = Session()
```

```
q1 = session.query(Departments).filter(Departments.dep_id==6)
```

```
q1.update({Departments.dep_name: '运维部'})
```

```
session.commit()
```

```
session.close()
```



## 更新数据（续1）

- 通过会话的字段赋值更新

```
from myorm import Session, Departments
```

```
session = Session()
```

```
q2 = session.query(Departments).get(1) # get(1)查询主键是1的记录
```

```
q2.dep_name = '开发部'
```

```
session.commit()
```

```
session.close()
```





# 删除记录

- 通过会话的delete()方法进行记录删除

```
from myorm import Session, Departments
```

```
session = Session()  
q1 = session.query(Departments).get(7)  
session.delete(q1)  
session.commit()  
session.close()
```



# 总结和答疑

---