课件：Python与数据库交互

python链接mysql

描述： 使用`pymysql`模块连接到MySQL数据库，并执行一些基本的数据操作

链接Demo：

准备工作：

> 首先, 导入了`pymysql`模块后，这是一个Python用于连接和操作MySQL数据库的库。
>
> #### 安装
>
> `pip install pymysql`
>
> 直接用pip安装默认是从国外仓库下载包，如果报错，可以选择换源（换中国镜像）
> 换源语法： `pip install pymysql -i 镜像源`
>
> 清华镜像源：https://pypi.tuna.tsinghua.edu.cn/simple
>
> 阿里云：https://mirrors.aliyun.com/pypi/simple/
>
> 实操举例： `pip install pymysql -i https://pypi.tuna.tsinghua.edu.cn/simple`

Python代码：

> 接下来，定义一个数据库配置字典`db_config`，其中包含了连接MySQL数据库所需的信息，包括主机名、端口号、用户名、密码、数据库名和字符集等。

```
参数解析：
 主机名（host）【str】：127.0.0.1，表示本地主机。(主机名 - ip : 指的是计算机的唯一标识符号)
 端口号（port）【int】：MySQL数据库的默认端口号3306。 (端口号： 程序运行后占用的数据通道)
 用户名（user）【str】：            (连接数据库时使用的用户名。)
 密码（password）【str】：         (连接数据库时使用的密码。)
 数据库名（db）【str】：            (连接到的数据库的名称。)
 字符集（charset）【str】：   utf8     (与数据库通信时使用的字符集。)
```

```python
import pymysql
# db_config  -> 数据库配置（Database Configuration）。数据库配置是用来连接和配置数据库的信息，包括数据库的主机名、端口号、用户名、密码等。
db_config = {
    'host': '127.0.0.1',
    'port': 3306,
    'user': '你自己创建的mysql账号',
    'password': '你自己创建的账号的密码',
    'db': '数据库名',
    'charset': 'utf8'
}
```

然后，使用`pymysql.connect()`函数根据数据库配置创建一个数据库连接对象`conn`，并使用该连接对象创建一个游标对象`cur`。

游标对象（Cursor Object）是数据库连接中的一种机制，用于在数据库结果集中进行导航和操作。它允许应用程序在结果集中逐行或批量获取数据，并对数据进行增、删、改和查询等操作

```python
# 关键字参数解包:使用 ** 时，它可以将一个字典中的键值对作为关键字参数传递给函数。
conn = pymysql.connect(**db_config)  # 获取链接对象
cur = conn.cursor()     # 游标对象
```

```
# 连接数据库的时候常见的报错
# 通常发生在配置数据库信息的时候,端口号不是整数类型
# ValueError: port should be of type int
# MySQL服务没有开启：
# pymysql.err.OperationalError: (2003, "Can't connect to MySQL server on '127.0.0.1' ([WinError 10061] 由于目标计算机积极拒绝，无法连接。)")
# 账号密码错误：
# pymysql.err.OperationalError: (1045, "Access denied for user 'buyuPY35'@'localhost' (using password: YES)")
```

进行数据库操作(**查询操作**)：

```
# 先在链接的库中创建一张表格备用
CREATE TABLE Users (
user_id INT PRIMARY KEY AUTO_INCREMENT,  # 用户ID，主键，自增*
username VARCHAR(50) NOT NULL unique key,            # 用户名，不允许为空，并且不能重复
password VARCHAR(255) NOT NULL,           # 密码，不允许为空*
isRoot VARCHAR(255) NOT NULL DEFAULT 0,           # 管理员状态，默认为0(0代表非管理员，1代表管理员)*
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, # 创建时间，默认值为当前时间戳*
```

```
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
     # 更新时间，默认值为当前时间戳，并在更新时自动更新*
    );

    # 新增一条管理员数据
    insert into Users(username,password,isRoot) values('root','root',1);

    # 假设这个表格是在用户管理系统中进行使用的，然后用户管理系统在进行程序设计的时候，规定了只有一个管理员账号，所以接下来的所有用户新增都是默认新增普通用户。

    # 新增普通用户数据(不需要插入管理员状态)
    insert into Users(username,password) values('不渝','1024')

    select * from Users;
```

python链接上MySQL操作示例代码：

```
# 如何用python链接上MySQL

# 一、导入 pymysql
import pymysql

# 二、定义一个数据库配置字典`db_config`，其中包含了连接MySQL数据库所需的信息
db_config = {
    'host' : '127.0.0.1',    # 主机名，由于目前链接的是电脑本地中的数据库，所以主机写127.0.0.1，这个地址是当前电脑的ip地址
    'port' : 3306,      # 端口号,程序运行后占用的数据通道
    'user' : 'root',      # 连接数据库时使用的用户名。
    'password' : 'qwe123',      # 连接密码
    'db' : 'mysql_six',      # 要连接的库
    'charset' : 'utf8'      # 字符集
}

# 三、用到以上的配置信息，进行数据库连接
# 关键字参数解包：试用**,可以将字典中的键值对都取出来作为函数的关键字参数
# 关键字参数，讲函数的时候有讲到，当时是讲了关键字不定长参数
conn = pymysql.connect(**db_config)      # 链接数据库，获取一个数据库连接对象
cur = conn.cursor()        # 获取专门用来操作数据库数据的对象,我们称呼这个对象为游标
print(conn)
print(cur)

# 四、对数据库中的表格数据进行增删改查操作
# 测试这个数据库连接后能不能对数据库中的数据进行查询

# 查询用户表功能函数
def selectUsers():
    # 1、写sql语句
    selectUsers_Sql = 'select * from Users;'
    # 2、通过python代码执行sql语句
    x = cur.execute(selectUsers_Sql)      # .execute() 是游标对象中用于执行sql的函数，他会返回一个执行影响值
    # print(x)
    # 3、查看查询结果
    userList = cur.fetchall()    # .fetchall()    用于获取所有查询到的数据
    # print(userList)
```

```
    return userList


# 新增(由于程序设定管理员账户不能由用户创建，所以这新增的都是普通用户)
def insert():
    try:
        username = input("请输入用户名：")
        password = input("请输入用户密码：")
        insertUsers_sql = f"insert into Users(username,password)
values('{username}','{password}');"
        # print(insertUsers_sql)
        x = cur.execute(insertUsers_sql)
        # print(x)
    except Exception as e:
        print("新增用户异常！ 当前用户名已存在")
        # 为了确保数据的安全性，一致性，完整性，我们可以在报错的时候，进行事务回滚，说白话就
是，撤销刚刚的操作，将数据库回退到操作前状态
        conn.rollback()      # 事务回滚
    else:       # 这里是代码没有报错会执行的
        # 为了确保数据的安全性，一致性，完整性，我们可以在操作正常完成无误后进行事务提交
        conn.commit()       # 事务提交
        print("恭喜账户创建成功！")
```

> 注意，这只是一个示例代码，在后期实际项目中，可以根据自己的实际需求和数据表结构进行相应的修改和扩展。
>
> 在实际应用中，还需要处理异常、使用参数化查询等来增加代码的健壮性和安全性。

用户管理-登陆注册小程序：

```
# 如何用python链接上MySQL

# 一、导入 pymysql
import pymysql

# 二、定义一个数据库配置字典`db_config`，其中包含了连接MySQL数据库所需的信息
db_config = {
    'host' : '127.0.0.1',    # 主机名，由于目前链接的是电脑本地中的数据库，所以主机写
127.0.0.1，这个地址是当前电脑的ip地址
    'port' : 3306,      # 端口号,程序运行后占用的数据通道
    'user' : 'root',      # 连接数据库时使用的用户名。
    'password' : 'qwe123',      # 连接密码
    'db' : 'mysql_six',      # 要连接的库
    'charset' : 'utf8'      # 字符集
}

# 三、用到以上的配置信息，进行数据库连接
# 关键字参数解包：试用**,可以将字典中的键值对都取出来作为函数的关键字参数
# 关键字参数，讲函数的时候有讲到，当时是讲了关键字不定长参数
conn = pymysql.connect(**db_config)      # 链接数据库，获取一个数据库连接对象
cur = conn.cursor()        # 获取专门用来操作数据库数据的对象,我们称呼这个对象为游标
print(conn)
print(cur)
```

```python
# 四、对数据库中的表格数据进行增删改查操作
# 测试这个数据库连接后能不能对数据库中的数据进行查询

# 查询用户表功能函数
def selectUsers():
    # 1、写sql语句
    selectUsers_Sql = 'select * from Users;'
    # 2、通过python代码执行sql语句
    x = cur.execute(selectUsers_Sql)     # .execute() 是游标对象中用于执行sql的函数，他会返回一个执行影响值
    # print(x)
    # 3、查看查询结果
    userList = cur.fetchall()   # .fetchall()   用于获取所有查询到的数据
    # print(userList)
    return userList


# 新增(由于程序设定管理员账户不能由用户创建，所以这新增的都是普通用户)
def insert():
    try:
        username = input("请输入用户名：")
        password = input("请输入用户密码：")
        insertUsers_sql = f"insert into Users(username,password) values('{username}','{password}');"
        # print(insertUsers_sql)
        x = cur.execute(insertUsers_sql)
        # print(x)
    except Exception as e:
        print("新增用户异常！ 当前用户名已存在")
        # 为了确保数据的安全性，一致性，完整性，我们可以在报错的时候，进行事务回滚，说白话就是，撤销刚刚的操作，将数据库回退到操作前状态
        conn.rollback()     # 事务回滚
    else:       # 这里是代码没有报错会执行的
        # 为了确保数据的安全性，一致性，完整性，我们可以在操作正常完成无误后进行事务提交
        conn.commit()       # 事务提交
        print("恭喜账户创建成功！")

# 用户管理系统-登陆注册功能
# 1、启动程序后，来到欢迎界面，用户可以选择注册、登陆
# 2、如果用户选择注册,则实现新增用户功能，然后退回欢迎界面
# 3、如果用户选择登陆，则进行权限校验，如果登陆成功，则提示欢迎后退出程序，如果登陆不成功则继续进行选择操作

# 欢迎界面函数
def HY():
    print("------------欢迎来到xx用户管理系统---------------")
    print("                  选项1:注册")
    print("                  选项2:登陆")
    c = input("    请输入你要进行操作选项的编号>>")
    return  c

# 解决登陆成功后，循环停止的问题：
hy_bool = True
while hy_bool:
```

```
    c = HY()
    if c=="1":
        print("即将进行注册")
        print("    -----欢迎来到xx用户管理系统注册界面----- ")
        # 调用新增用户的函数
        insert()
    elif c == "2":
        print("进行登陆验证！")
        # 输入登陆账户密码
        user_name = input("请输入用户名：")
        pass_word = input("请输入用户密码：")
        # 用户输入的密码是要与数据库中的数据进行校验
        li = selectUsers()       # 调用已经写好的，查询用户表功能函数，获取到用户表中所有用户信息
        for user_msg in li:        # 用循环遍历 用户信息，每遍历一个进行打印，
            # 遍历后发现每个用户信息是存储在元组中，元组是序列类型数据，可以下标取值
            if user_name==user_msg[1] and pass_word==user_msg[2]:
                print("登陆成功")
                # 由于登陆成功了之后，我们就不需要再循环是否要登陆还是注册了，所以我们就不需继续这个while循环了
                hy_bool = False
                break     # 结束循环
            print(user_msg)
        else:
            # Python 中，无论是 while 循环还是 for 循环，其后都可以紧跟着一个 else 代码块，
            # 它的作用是当循环条件为 False 跳出循环时，程序会最先执行 else 代码块中的代码。
            print("登陆失败！")
```

mysql事务

在数据库管理中，事务（Transaction）是一组SQL操作，它们要么全部成功执行，要么全部失败，没有中间状态。事务的目的是确保数据库的一致性和可靠性，保证数据的完整性。

事务通常具有以下四个特性，通常称为ACID特性：

原子性（Atomicity）：

一个事务（transaction）中的操作被视为一个原子单元，要么全部执行成功，要么全部不执行。如果事务中的任何操作失败，整个事务将被回滚到事务开始前的状态，不会留下部分完成的操作。原子性确保了数据库的一致性。

一致性（Consistency）：

在事务开始之前和结束之后，数据库的完整性约束没有被破坏。这意味着事务操作必须使数据库从一个一致的状态转移到另一个一致的状态。例如，如果数据库中定义了某个属性的取值范围为1到100，那么事务中的任何操作都不能导致该属性的值超出这个范围。

隔离性（Isolation）：

多个并发事务同时执行时，每个事务都应该被隔离开，互相之间不会产生干扰。隔离性确保了每个事务在逻辑上独立运行，就像是在系统中没有其他事务同时执行一样。隔离级别定义了事务之间的隔离程度，包括读未提交、读提交、可重复读和串行化。

持久性（Durability）：

一旦事务提交成功，对数据库的修改就是永久的，即使发生系统故障或崩溃，修改的数据也会被持久保存在数据库中。持久性保证了事务的结果不会丢失。

这些特性共同确保了关系型数据库的可靠性、一致性和持久性。通过遵循ACID特性，关系型数据库能够处理事务，并保证数据的完整性和一致性，从而满足许多应用场景中的数据管理需求。

数据库演示（了解）

```
create table new_tb_name(id int, name varchar(20), age int);

begin; #开启事物
select * from new_tb_name;
insert into new_tb_name values (3,'hello',18);
commit;  #这个语句提交了事务，将当前事务所做的更改永久保存下来。因此，前面的 insert 语句中插入的记录现在已经持久化到表中。

表 new_tb_name 的状态如下:
    +----+-------+-----+
    | id | name  | age |
    +----+-------+-----+
    | 3  | hello | 18  |
    +----+-------+-----+

begin;
insert into new_tb_name values (5,'jack',11);
insert into new_tb_name values (6,'jack',110);
rollback; #撤销全部 【 这个语句回滚了当前事务，取消了事务中所做的所有更改。因此，该事务中的两个 insert 语句都被撤销了。】

select * from new_tb_name;

在回滚之后，表 new_tb_name #仍然只有一条记录:
    +----+-------+-----+
    | id | name  | age |
    +----+-------+-----+
    | 3  | hello | 18  |
    +----+-------+-----+
```

事务拓展（不要求掌握）

【隔离级别定义了事务之间的隔离程度，包括读未提交、读提交、可重复读和串行化。】

读未提交（Read uncommitted） 顾名思义，就是一个事务可以读取另一个未提交事务的数据。

事例：

老板要给程序员发工资，程序员的工资是3.6万/月。但是财务发工资条时不小心按错了数字，按成3.9万/月，该工资条已经发到程序员邮箱，但是事务还没有提交（工资未发放），就在这时，程序员去查看自己这个月的工资，发现比往常多了3千元，以为涨工资了非常高兴。但是财务及时发现了不对，马上回滚差点就提交了的事务，将数字改成3.6万再提交（工资发放）

分析：

实际程序员这个月的工资还是3.6万，但是程序员看到的是3.9万。他看到的是公司还没提交事务时的数据。这就是脏读。

那怎么解决脏读呢？ Read committed！读提交，能解决脏读问题。

读提交（read committed）顾名思义，就是一个事务要等另一个事务提交后才能读取数据。
事例：

程序员拿着信用卡去享受生活（卡里当然是只有3.6万），当他埋单时（程序员事务开启），收费系统事先检测到他的卡里有3.6万，就在这个时候！！程序员的妻子要把钱全部转出充当家用，并提交。当收费系统准备扣款时，再检测卡里的金额，发现已经没钱了（第二次检测金额当然要等待妻子转出金额事务提交完）。程序员就会很郁闷，明明卡里是有钱的...
分析：

这就是读提交，若有事务对数据进行更新（UPDATE）操作时，读操作事务要等待这个更新操作事务提交后才能读取数据，可以解决脏读问题。但在这个事例中，出现了一个事务范围内两个相同的查询却返回了不同数据，这就是不可重复读。

那怎么解决可能的不可重复读问题？Repeatable read ！

可重复读（repeatable read）就是在开始读取数据（事务开启）时，不再允许修改操作
事例：

程序员拿着信用卡去享受生活（卡里当然是只有3.6万），当他埋单时（事务开启，不允许其他事务的UPDATE修改操作），收费系统事先检测到他的卡里有3.6万。这个时候他的妻子不能转出金额了。接下来收费系统就可以扣款了。
分析：

重复读可以解决不可重复读问题。写到这里，应该明白的一点就是，不可重复读对应的是修改，即UPDATE操作。但是可能还会有幻读问题。因为幻读问题对应的是插入INSERT操作，而不是UPDATE操作。

什么时候会出现幻读?
事例：

程序员某一天去消费，花了2千元，然后他的妻子去查看他今天的消费记录（全表扫描FTS，妻子事务开启），看到确实是花了2千元，就在这个时候，程序员花了1万买了一部电脑，即新增INSERT了一条消费记录，并提交。当妻子打印程序员的消费记录清单时（妻子事务提交），发现花了1.2万元，似乎出现了幻觉，这就是幻读。

那怎么解决幻读问题？Serializable！

串行化（Serializable）Serializable 是最高的事务隔离级别，在该级别下，事务串行化顺序执行，可以避免脏读、不可重复读与幻读。但是这种事务隔离级别效率低下，比较耗数据库性能，一般不使用。
值得一提的是：大多数数据库默认的事务隔离级别是Read committed，比如Sql Server , Oracle。
Mysql的默认隔离级别是Repeatable read。

Json（建议拓展）

JSON（JavaScript Object Notation）是一种轻量级的数据交换格式。它以易于阅读和编写的文本格式表示结构化数据，通常用于将数据从服务器传输到客户端，或在应用程序之间进行数据交换。

JSON采用了一种简洁的键值对的方式来表示数据。它由两种基本结构组成：

1. 对象（Object）：由大括号 `{}` 包围，表示无序的键值对集合。每个键值对中，键（key）是一个字符串，值（value）可以是字符串、数字、布尔值、对象、数组或null。键和值之间使用冒号 `:` 分隔，键值对之间使用逗号 `,` 分隔。

   示例：

   ```
   {
     "name": "John",
     "age": 30,
     "isStudent": false,
     "address": {
       "city": "New York",
       "country": "USA"
     },
     "hobbies": ["reading", "playing guitar", "traveling"]
   }
   ```

2. 数组（Array）：由方括号 `[]` 包围，表示有序的值的集合。数组中的值可以是字符串、数字、布尔值、对象、数组或null。值之间使用逗号 `,` 分隔。

   示例：

   ```
   [
     "apple",
     "banana",
     "orange"
   ]
   ```

JSON的设计简洁且易于解析和生成，广泛应用于Web开发、API通信、配置文件等场景。各种编程语言都提供了处理JSON数据的库或内置函数，使得在应用程序中解析和生成JSON变得非常方便。以下是Python中的json

```
json 跟我们的字典很像
经常提到 但是一直没有给你们去解释   -> 文本格式
前后端交互既基本都是以json传输的
# 不同语言对话

import json   # 导入了json模块

dict_a = {'name':'qiye','age':None,'isMan':True}    # 定义了一个字典dict_a，其中包含了不同类型的键值对
json_a = json.dumps(dict_a)     # json.dumps()函数将字典转换为JSON格式的字符串json_a
print(type(json_a),json_a)     # 查看json_a的类型和内容

new_dict_a = json.loads(json_a)     # 使用json.loads()函数将JSON字符串json_a解析为Python字典对象new_dict_a
print(type(new_dict_a),new_dict_a)      # 通过print()函数打印出new_dict_a的类型和内容。
```

```python
# 使用json.dump()函数将字典dict_a以JSON格式写入到名为tt.json的文件中
with open('tt.json','w') as f:
    json.dump(dict_a,f)     # json.dump()函数将字典序列化为JSON字符串，并将其写入文件。

# 使用json.load()函数从文件中读取JSON数据并将其解析为Python对象。
with open('tt.json','r') as f1:
    # json.load()函数从文件中读取JSON字符串，并将其解析为Python数据类型。
    new_dict_b = json.load(f1)     # 将文件tt.json中的JSON数据读取为字典new_dict_b
    print(new_dict_b)
```

代码展示了使用json模块进行JSON数据的序列化和反序列化，以及读写JSON文件的基本操作。

作业

1、做好前提准备工作，定义一个数据库配置字典`db_config`，其中包含了连接MySQL数据库所需的信息，包括主机名、端口号、用户名、密码、数据库名和字符集等。
2、创建数据库连接
3、结合Python基础语法中的异常处理和mysql事务，完成课堂案例。
'''作业交执行结果截图'''
4、拓展一下事务的概念
5、拓展一下json数据

注意：在pycharm中进行完练习后可以，防止资源浪费
```
cur.close() # 关闭游标。
conn.close() # 关闭连接 connection
```
终端里面不用加