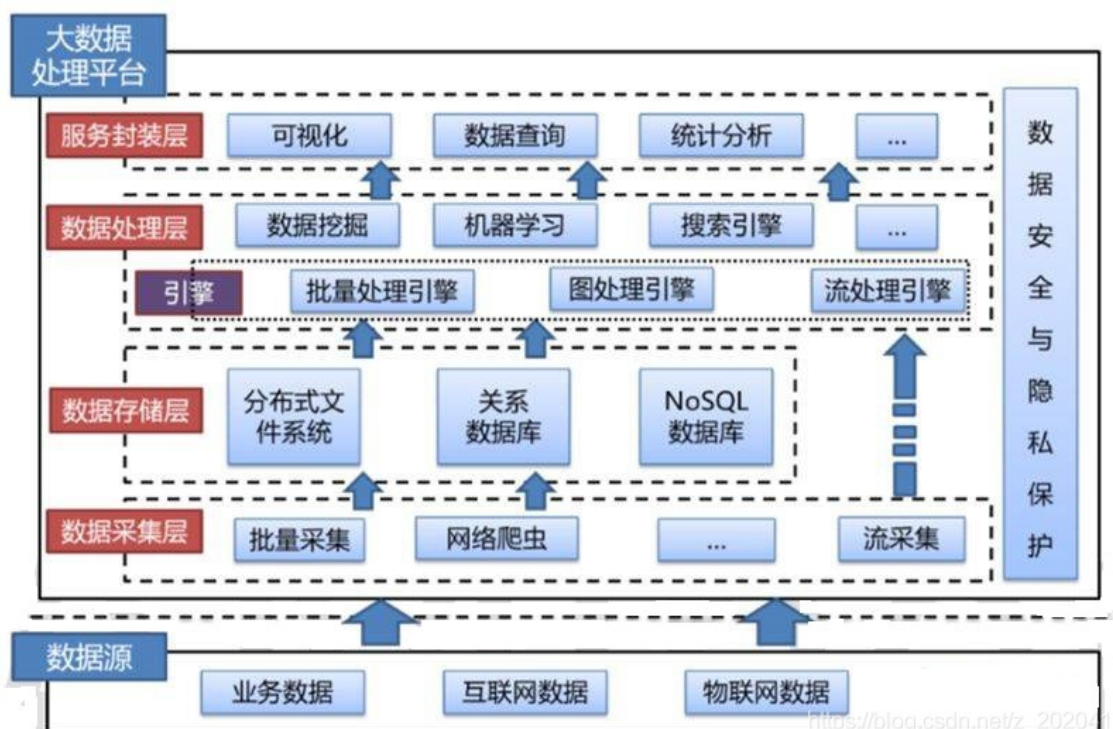


@[toc]

# 数据处理

- 数据处理概述

数据处理的基本目的是从大量的、可能是杂乱无章的、难以理解的数据中抽取并推导出对于某些特定的人们来说是有价值、有意义的数据。当下数据处理贯穿于社会生产和社会生活的各个领域。数据处理技术的发展及其应用的广度和深度，极大地影响了人类社会发展的进程。数据处理也是大数据，数据分析等后续科学的基本环节。



- 基本概念

- 数据：能够输入到计算机中并被识别处理的信息集合。
- 大数据：是指无法在一定时间范围内用一定工具进行捕捉、管理和处理的数据集合，是海量、高增长率和多样化的信息资产。

- 数据存储阶段

- 人工管理阶段：人为管理，没有固定的格式和存储方法，容易混乱。
- 文件管理阶段：数据可以长期保存，存储数据量大，使用简单。
- 数据库管理阶段：高效，可以存储更大量数据，便于管理，更加专业。

## 1. 文件处理

## 1.1 引入

- 什么是文件

文件是保存在持久化存储设备(硬盘、U盘、光盘..)上的一段数据，一个文本，一个py文件，一张图片，一段视…… 这些都是文件。

- 文件分类

- 文本文件：打开后会自动解码为字符，如txt文件，word文件，py程序文件。
- 二进制文件：内部编码为二进制码，无法通过文字编码解析，如压缩包，音频，视频，图片等。

- 字节串类型

- 概念：在python3中引入了字节串的概念，与str不同，字节串以不可变字节序列值表达数据，更方便用来处理二进制数据。
- 字符串与字节串相互转化方法

```
1 - 普通的英文字符字符串常量可以在前面加b转换为字节串，例如：b'hello'
2 - 变量或者包含非英文字符的字符串转换为字节串方法：str.encode()
3 - 字节串转换为字符串方法：bytes.decode()
4
5 注意：python字符串用来表达utf8字符，因为并不是所有二进制内容都可以转化为utf8字
   符，所以不是所有字节串都能转化为字符串，但是所有字符串都能转化成二进制，所以所有字
   符串都能转换为字节串。
6 一个汉字3个字节。
```

## 1.2 文件读写操作

使用程序操作文件，无外乎对文件进行读或者写

- 读：即从文件中获取内容
- 写：即修改文件中的内容

对文件实现读写的基本操作步骤为：打开文件，读写文件，关闭文件。

### 1.2.1 打开文件

```
1 file_object = open(file_name, access_mode='r', buffering=-1)
2 功能：打开一个文件，返回一个文件对象。
3 参数：file_name 文件名：
4       access_mode 打开文件的方式,如果不写默认为'r'
5       buffering 1表示有行缓冲，默认则表示使用系统默认提供的缓冲机制。
6 返回值：成功返回文件操作对象。
```

打开模式	效果
r	以读方式打开，文件必须存在
w	以写方式打开，文件不存在则创建，存在清空原有内容
a	以追加模式打开，文件不存在则创建，存在则继续进行写操作
r+	以读写模式打开 文件必须存在
w+	以读写模式打开文件，不存在则创建，存在清空原有内容
a+	追加并可读模式，文件不存在则创建，存在则继续进行写操作
rb	以二进制读模式打开 同r
wb	以二进制写模式打开 同w
ab	以二进制追加模式打开 同a
rb+	以二进制读写模式打开 同r+
wb+	以二进制读写模式打开 同w+
ab+	以二进制读写模式打开 同a+

注意：

1. 以二进制方式打开文件，读取内容为字节串，写入也需要写入字节串
2. 无论什么文件都可以使用二进制方式打开，但是二进制文件则不能以文本方式打开，否则后续读写会报错。

## 1.2.2 读取文件

- 方法1

```

1 read([size])
2 功能： 来直接读取文件中字符。
3 参数： 如果没有给定size参数（默认值为-1）或者size值为负，文件将被读取直至末尾，给定size最多读取给定数目个字符（字节）。
4 返回值： 返回读取到的内容 ---字符串

```

注意：文件过大时候不建议直接读取到文件结尾，读到文件结尾会返回空字符串或空字节串。

```

1 # 大文件循环读取文件
2 while True:
3     data = f.read(100) # 每次读取100个字节，读到文件结尾后会返回空字符串
4     if not data:
5         break
6     print(data)

```

- 方法2

```
1 readline([size])
2 功能： 用来读取文件中一行
3 参数： 如果没有给定size参数（默认值为-1）或者size值为负，表示读取一行，给定size表示最多读取指定的字符（字节）。
4 返回值： 返回读取到的内容 ---字符串
```

- 方法3

```
1 readlines([sizeint])
2 功能： 读取文件中的每一行作为列表中的一行
3 参数： 如果没有给定size参数（默认值为-1）或者size值为负，文件将被读取直至末尾，给定size表示读取到size字符所在行为止。
4 返回值： 返回读取到的内容列表 ---列表
```

- 方法4

```
1 # 文件对象本身也是一个可迭代对象，在for循环中可以迭代文件的每一行。
2
3 for line in f:
4     print(line)
```

### 1.2.3 写入文件

- 方法1

```
1 write(data)
2 功能： 把文本数据或二进制数据块的字符串写入到文件中去
3 参数： 要写入的内容
4 返回值： 写入的字符个数
```

注意： 如果需要换行要自己在写入内容中添加\n

- 方法2

```
1 writelines(str_list)
2 功能： 接受一个字符串列表作为参数，将它们写入文件。
3 参数： 要写入的内容列表
```

读-写： 4-2

### 1.2.4 关闭文件

打开一个文件后我们就可以通过文件对象对文件进行操作了，当操作结束后可以关闭文件操作

- 方法

```
1 file_object.close()
```

- 好处

1. 可以销毁对象节省资源，（当然如果不关闭程序结束后对象也会被销毁）。
2. 防止后面对这个对象的误操作。

1.2.5 with操作

python中的with语句也可以用于访问文件，在语句块结束后会自动释放资源。

- with语句格式

```
1 with context_expression [as obj]:
2     with-body
```

- with访问文件

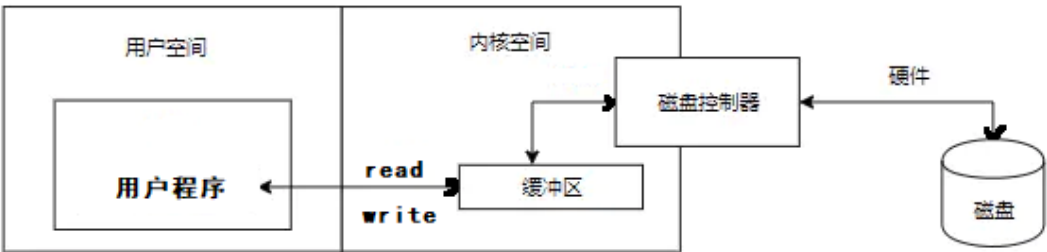
```
1 with open('file','r+') as f:
2     f.read()
```

注意： with语句块结束后会自动释放f所以不再需要close()。

1.2.6 缓冲区

- 定义

系统自动的在内存中为每一个正在使用的文件开辟一个空间，在对文件读写时都是先将文件内容加载到缓冲区，再进行读写。



[https://blog.csdn.net/z\\_202041](https://blog.csdn.net/z_202041)

- 作用
  - 1. 减少和磁盘的交互次数，保护磁盘。
  - 2. 提高了对文件的读写效率。
- 缓冲区设置

类型	设置方法	注意事项
系统自定义	buffering=-1	
行缓冲	buffering=1	当遇到\n时刷新缓冲
指定缓冲区大小	buffering>1	必须以二进制方式打开

- 刷新缓冲区条件
  - 1. 缓冲区被写满

2. 程序执行结束或者文件对象被关闭
3. 程序中调用flush()函数

```
1 file_obj.flush()
```

### 1.2.7 文件偏移量

- 定义

打开一个文件进行操作时系统会自动生成一个记录，记录每次读写操作时所处的文件位置，每次文件的读写操作都是从这个位置开始进行的。

注意：

1. r或者w方式打开，文件偏移量在文件开始位置
2. a方式打开，文件偏移量在文件结尾位置

- 文件偏移量控制

```
1 tell()
2 功能：获取文件偏移量大小
3 返回值：文件偏移量
```

```
1 seek(offset[,whence])
2 功能：移动文件偏移量位置
3 参数：offset 代表相对于某个位置移动的字节数。负数表示向前移动，正数表示向后移动。
4         whence是基准位置的默认值为 0，代表从文件开头算起，1代表从当前位置算起，2 代表从文件末尾算起。
```

注意：必须以二进制方式打开文件时，基准位置才能是1或者2

## 1.3 os模块

os模块是Python标准库模块，包含了大量的文件处理函数。

- 获取文件大小

```
1 os.path.getsize(file)
2 功能： 获取文件大小
3 参数： 指定文件
4 返回值： 文件大小
```

- 查看文件列表

```
1 os.listdir(dir)
2 功能： 查看文件列表
3 参数： 指定目录
4 返回值： 目录中的文件名列表
```

- 查看文件是否存在

```
1 os.path.exists(file)
2 功能： 查看文件是否存在
3 参数： 指定文件
4 返回值： 存在返回True，不存在返回False
```

- 判断文件类型

```
1 os.path.isfile(file)
2 功能： 判断文件类型
3 参数： 指定文件
4 返回值： 普通文件返回True，否则返回False
```

- 删除文件

```
1 os.remove(file)
2 功能： 删除文件
3 参数： 指定文件
```

```
1 """
2 os模块处理文件
3 """
4
5 import os
6
7 print(os.path.getsize('my.log')) # 获取文件大小
8
9 print(os.listdir('.')) # 查看一个目录中有什么文件
10
11 print(os.path.exists('abc')) # 查看一个文件是否存在
12
13 print(os.path.isfile('my.log')) # 查看文件是否为普通文件
14
15 os.remove('my.log') # 删除一个文件
16 -----
17 0
18 ['regex2.py', 'exercise_2.py', '4.txt', 'regex3.py', 'exercise_1.py',
19  'my.log', 'log.txt', 'file.py', 'regex1.py']
20 False
21 True
```

```
1 """
2 重点代码
3 练习4： 文件的拷贝
4         有一个文件 timg.jpeg,将其拷贝一份到主目录下，命名为 mm.jpg
5
6         从源文件读取，写入到目标文件
7
8 思路： 从timg.jpeg中读取内容，原样写入到mm.jpg
9 """
10
11 fr = open('timg.jpeg','rb') # 源文件
12 fw = open('/home/tarena/mm.jpg','wb') # 新文件
```

```
13
14 while True:
15     data = fr.read(1024)
16     # 读到文件结尾返回空字符串
17     if not data:
18         break
19     fw.write(data)
20
21 fr.close()
22 fw.close()
23
```

## 2. 正则表达式

### 2.1 概述

- 学习动机
  1. 文本数据处理已经成为常见的编程工作之一
  2. 对文本内容的**搜索，定位，提取**是逻辑比较复杂的工作
  3. 为了快速方便的解决上述问题，产生了**正则表达式技术**
- 定义

即**文本的高级匹配模式**，其本质是由一系列字符和特殊符号构成的字串，这个字串即正则表达式。

- 原理

通过普通字符和有特定含义的字符，来组成字符串，用以描述一定的字符串规则，比如：重复，位置等，来表达某类特定的字符串，进而匹配。

- 学习目标
  1. 熟练掌握正则表达式元字符
  2. 能够读懂常用正则表达式，编辑简单的正则规则
  3. 能够熟练使用re模块操作正则表达式

### 2.2 元字符使用

- 普通字符

匹配规则：每个普通字符匹配其对应的字符

```
1 e.g.
2 In : re.findall('ab',"abcdefabcd")
3 Out: ['ab', 'ab']
4 In [20]: re.findall("ab","afggasabfdfab")
5 Out[20]: ['ab', 'ab']
```

注意：正则表达式在python中也可以匹配中文

- 或关系



元字符: `|`

匹配规则: 匹配 | 两侧任意的正则表达式即可

```
1 e.g.  
2 In : re.findall('com|cn','www.baidu.com/www.tmooc.cn')  
3 Out: ['com', 'cn']  
4
```

- 匹配单个字符

元字符: `.`

匹配规则: 匹配除换行外的任意一个字符

```
1 e.g.  
2 In : re.findall('张.丰','张三丰,张四丰,张五丰')  
3 Out: ['张三丰', '张四丰', '张五丰']  
4
```

- 匹配字符集

元字符: `[]` 字符集

匹配规则: 匹配字符集中的任意一个字符

表达形式:

[abc#!好] 表示 [] 中的任意一个字符  
[0-9],[a-z],[A-Z] 表示区间内的任意一个字符  
[#?0-9a-z] 混合书写, 一般区间表达写在后面

```
1 e.g.  
2 In : re.findall('[aeiou]','How are you!')  
3 Out: ['o', 'a', 'e', 'o', 'u']
```

- 匹配字符集反集

元字符: `^` 字符集

匹配规则: 匹配除了字符集以外的任意一个字符

```
1 e.g.  
2 In : re.findall('[^0-9]','use 007 port')  
3 Out: ['u', 's', 'e', ' ', ' ', ' ', 'p', 'o', 'r', 't']
```

- 匹配字符串开始位置

元字符: `^`

匹配规则: 匹配目标字符串的开头位置

```
1 e.g.  
2 In : re.findall('^Jame', "Jame,hello")  
3 Out: ['Jame']
```

- 匹配字符串的结束位置

元字符: `$`

匹配规则: 匹配目标字符串的结尾位置

```
1 e.g.  
2 In : re.findall('Jame$', "Hi, Jame")  
3 Out: ['Jame']
```

规则技巧: `^` 和 `$` 必然出现在正则表达式的开头和结尾处。如果两者同时出现, 则中间的部分必须匹配整个目标字符串的全部内容。

- 匹配字符重复

元字符: `*`

匹配规则: 匹配前面的字符出现0次或多次

```
1 e.g.  
2 In : re.findall('wo*', "wooooo~~w!")  
3 Out: ['wooooo', 'w']
```

元字符: `+`

匹配规则: 匹配前面的字符出现1次或多次

```
1 e.g.  
2 In : re.findall('[A-Z][a-z]+', "Hello world")  
3 Out: ['Hello', 'world']
```

元字符: `?`

匹配规则: 匹配前面的字符出现0次或1次

```
1 e.g. 匹配整数  
2 In [28]: re.findall('-?[0-9]+', "Jame,age:18, -26")  
3 Out[28]: ['18', '-26']
```

元字符: `{n}`

匹配规则: 匹配前面的字符出现n次

```
1 e.g. 匹配手机号码
2 In : re.findall('1[0-9]{10}','Jame:13886495728')
3 Out: ['13886495728']
4
```

元字符: `{m,n}`

匹配规则: 匹配前面的字符出现m-n次

```
1 e.g. 匹配qq号
2 In : re.findall('[1-9][0-9]{5,10}','Baron:1259296994')
3 Out: ['1259296994']
```

- 匹配任意 (非) 数字字符

元字符: `\d` `\D`

匹配规则: `\d` 匹配任意数字字符, `\D` 匹配任意非数字字符

```
1 e.g. 匹配端口
2 In : re.findall('\d{1,5}','Mysql: 3306, http:80')
3 Out: ['3306', '80']
```

- 匹配任意 (非) 普通字符

元字符: `\w`

匹配规则: `\w` 匹配普通字符, `\W` 匹配非普通字符

说明: 普通字符指数字, 字母, 下划线, 汉字。

```
1 e.g.
2 In : re.findall('\w+', 'server_port = 8888')
3 Out: ['server_port', '8888']
```

- 匹配任意 (非) 空字符

元字符: `\s` `\S`

匹配规则: `\s` 匹配空字符, `\S` 匹配非空字符

说明: 空字符指 空格 `\r` `\n` `\t` `\v` `\f` 字符

```
1 e.g.
2 In : re.findall('\w+\s+\w+', 'hello world')
3 Out: ['hello world']
```

- 匹配 (非) 单词的边界位置

元字符: `\b` `\B`

匹配规则: `\b` 表示单词边界, `\B` 表示非单词边界

说明: 单词边界指数字字母(汉字)下划线与其他字符的交界位置。

```
1 e.g.  
2 In : re.findall(r'\bis\b','This is a test.')  
3 Out: ['is']
```

注意: 当元字符符号与Python字符串中转义字符冲突的情况则需要使用r将正则表达式字符串声明为原始字符串, 如果不确定那些是Python字符串的转义字符, 则可以在所有正则表达式前加r。

类别	元字符
匹配字符	<code>.</code> <code>[...]</code> <code>^[...]</code> <code>\d</code> <code>\D</code> <code>\w</code> <code>\W</code> <code>\s</code> <code>\S</code>
匹配重复	<code>*</code> <code>+</code> <code>?</code> <code>{n}</code> <code>{m,n}</code>
匹配位置	<code>^</code> <code>\$</code> <code>\b</code> <code>\B</code>
其他	<code> </code> <code>()</code> <code>\</code>

## 2.3 匹配规则

### 2.3.1 特殊字符匹配

- 目的: 如果匹配的目标字符串中包含正则表达式特殊字符, 则在表达式中元字符就想表示其本身含义时就需要进行 `\` 处理。

```
1 特殊字符: . * + ? ^ $ [ ] ( ) { } | \
```

- 操作方法: 在正则表达式元字符前加 `\` 则元字符就是去其特殊含义, 就表示字符本身

```
1 e.g. 匹配特殊字符 . 时使用 \. 表示本身含义  
2 In : re.findall('-?\d+\.?\d*', '123,-123,1.23,-1.23')  
3 Out: ['123', '-123', '1.23', '-1.23']
```

### 2.3.2 贪婪模式和非贪婪模式

- 定义

贪婪模式: 默认情况下, 匹配重复的元字符总是尽可能多的向后匹配内容。比如: `* + ? {m,n}`

非贪婪模式(懒惰模式): 让匹配重复的元字符尽可能少的向后匹配内容。

- 贪婪模式转换为非贪婪模式

在对应的匹配重复的元字符后加 '?' 号即可

```
1 * -> *?
2 + -> +?
3 ? -> ??
4 {m,n} -> {m,n}?
```

```
1 e.g.
2 In : re.findall(r'\(.*?\)', "(abcd)efgh(higk)")
3 Out: ['(abcd)', '(higk)']
```

### 2.3.3 正则表达式分组

- 定义

在正则表达式中，以()建立正则表达式的内部分组，子组是正则表达式的一部分，可以作为内部整体操作对象。

- 作用：可以被作为整体操作，改变元字符的操作对象

```
1 e.g. 改变 +号 重复的对象
2 In : re.search(r'(ab)+', "ababababab").group()
3 Out: 'ababababab'
4
5 e.g. 改变 |号 操作对象
6 In : re.search(r'(王|李)\w{1,3}', "王者荣耀").group()
7 Out: '王者荣耀'
```

- 捕获组

捕获组本质也是一个子组，只不过拥有一个名称用以表达该子组的意义，这种有名称的子组即为捕获组。

格式：( ?P<name>pattern )

```
1 e.g. 给予组命名为 "pig"
2 In : re.search(r'(?P<pig>ab)+', "ababababab").group('pig')
3 Out: 'ab'
4 In [5]: re.search(r'(?P<pig>ab)+', "ababababab").group()
5 Out[5]: 'ababababab'
6
```

- 注意事项
- 一个正则表达式中可以包含多个子组
- 子组可以嵌套但是不宜结构过于复杂
- 子组序列号一般从外到内，从左到右计数

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-5xCph975-1610634103049)(./img/re.png)]

### 2.3.4 正则表达式匹配原则

1. 正确性,能够正确的匹配出目标字符串.
2. 排他性,除了目标字符串之外尽可能少的匹配其他内容.
3. 全面性,尽可能考虑到目标字符串的所有情况,不遗漏.

## 2.4 Python re模块使用

### 2.4.1 基础函数使用

---

```
1 regex = compile(pattern, flags = 0)
2 功能: 生产正则表达式对象
3 参数: pattern 正则表达式
4       flags 功能标志位, 扩展正则表达式的匹配
5 返回值: 正则表达式对象
```

---

```
1 re.findall(pattern, string, flags = 0)
2 功能: 根据正则表达式匹配目标字符串内容
3 参数: pattern 正则表达式
4       string 目标字符串
5       flags 功能标志位, 扩展正则表达式的匹配
6 返回值: 匹配到的内容列表, 如果正则表达式有子组则只能获取到子组对应的内容
```

---

```
1 regex.findall(string, pos, endpos)
2 功能: 根据正则表达式匹配目标字符串内容
3 参数: string 目标字符串
4       pos 截取目标字符串的开始匹配位置
5       endpos 截取目标字符串的结束匹配位置
6 返回值: 匹配到的内容列表, 如果正则表达式有子组则只能获取到子组对应的内容
```

---

```
1 re.split(pattern, string, max, flags = 0)
2 功能: 使用正则表达式匹配内容, 切割目标字符串
3 参数: pattern 正则表达式
4       string 目标字符串
5       max 最多切割几部分
6       flags 功能标志位, 扩展正则表达式的匹配
7 返回值: 切割后的内容列表
```

---

```
1 re.sub(pattern,replace,string,count,flags = 0)
2 功能：使用一个字符串替换正则表达式匹配到的内容
3 参数：pattern 正则表达式
4       replace 替换的字符串
5       string 目标字符串
6       count 最多替换几处,默认替换全部
7       flags 功能标志位,扩展正则表达式的匹配
8 返回值：替换后的字符串
```

```
1 """
2 re模块演示1
3 """
4 import re
5
6 # 目标字符串
7 s = "Alex:1996,Sunny:1998"
8 pattern = r"(\w+):(\d+)" # 正则表达式
9 # pattern = r"\w+:\d+" # 正则表达式
10
11 # re直接调用
12 l = re.findall(pattern,s)
13 print(l)
14
15 # 生成正则表达式对象
16 regex = re.compile(pattern)
17 l = regex.findall(s,0,13) # 目标字符串变为 s[0:13]
18 print(l)
19
20 # 使用正则表达式匹配内容分割字符串
21 l = re.split(r"\w+",s,2)
22 print(l)
23
24 # 替换目标字符串中匹配到的内容
25 s = re.sub(r"\w+", '##', s, 2)
26 print(s)
27 -----
28 [('Alex', '1996'), ('sunny', '1998')]
29 [('Alex', '1996')]
30 ['Alex', '1996', 'Sunny:1998']
31 Alex##1996##Sunny:1998
```

## 2.4.2 生成match对象

```
1 re.finditer(pattern,string,flags = 0)
2 功能：根据正则表达式匹配目标字符串内容
3 参数：pattern 正则表达式
4       string 目标字符串
5       flags 功能标志位,扩展正则表达式的匹配
6 返回值：匹配结果的迭代器
```

```
1 re.match(pattern,string,flags=0)
2 功能：匹配某个目标字符串开始位置
3 参数：pattern 正则
4       string 目标字符串
5 返回值：匹配内容match object 没有匹配到返回None
```

```
1 re.search(pattern,string,flags=0)
2 功能：匹配目标字符串第一个符合内容
3 参数：pattern 正则
4       string 目标字符串
5 返回值：匹配内容match object 没有匹配到返回None
```

```
1 """
2 re模块示例，生成match对象
3
4 一个match对象对应一处匹配内容
5 用于获取匹配到内容的详细信息
6 """
7
8 import re
9
10 s = "时，2020年春，AID 2002 班开课"
11 # s = "2020年春，AID 2002 班开课"
12
13 pattern = r"\d+"
14
15 # 匹配返回迭代器
16 it = re.finditer(pattern ,s)
17
18 for i in it:
19     print(i.group()) # 得到match对象
20
21 # 匹配开头位置
22 # obj = re.match(pattern,s)
23 # print(obj.group())
24
25 # 匹配第一处
26 obj = re.search(pattern,s)
27 print(obj.group())
28
29
```

### 2.4.3 match对象使用

- span() 获取匹配内容的起止位置，返回2元组
- groupdict() 获取捕获组字典，组名为键，对应内容为值
- groups() 获取子组对应内容，返回元组
- group(n = 0)

功能：获取match对象匹配内容

参数：默认为0表示获取整个match对象内容，如果是序列号或者组名则表示获取对应子组内容

返回值：匹配字符串



```

1  """
2  match对象使用
3  """
4
5  import re
6
7  s = "abcdefghi"
8  pattern = "(ab)cd(?P<dog>ef)"
9
10 obj = re.search(pattern,s) # 获取match对象
11
12 print(obj.span()) # s[0:6] 获取匹配到的内容在目标字符串中的位置
13 print(obj.groupdict()) # 获取捕获组字典 组名为键，匹配到的内容为值
14 print(obj.groups()) # 获取子组匹配到的内容
15 print(obj.group()) # 获取匹配内容
16 -----
17 (0, 6)
18 {'dog': 'ef'}
19 ('ab', 'ef')
20 abcdef

```

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-IEOqJG8n-1610634103056)(./img/re1.png)]

## 2.4.4 flags参数扩展

- 作用函数：re模块调用的匹配函数。如：re.compile,re.findall,re.search....
- 功能：扩展丰富正则表达式的匹配功能
- 常用flag

```

1  A == ASCII   元字符只能匹配ascii码
2
3  I == IGNORECASE  匹配忽略字母大小写
4
5  S == DOTALL   使 . 可以匹配换行
6
7  M == MULTILINE  使 ^  $可以匹配每一行的开头结尾位置

```

注意：同时使用多个flag，可以用竖线连接 flags = re.I | re.A

## 3. 数据库

## 3.1概述

- 数据存储

1. 人工管理阶段

缺点： 数据存储量有限，共享处理麻烦，操作容易混乱

2. 文件管理阶段 (.txt .doc .xls)

优点： 数据可以长期保存,可以存储大量的数据,使用简单。

缺点： 数据一致性差,数据查找修改不方便,数据冗余度可能比较大。

3. 数据库管理阶段

优点： 数据组织结构化降低了冗余度,提高了增删改查的效率,容易扩展,方便程序调用处理

缺点： 需要使用sql 或者其他特定的语句，相对比较专业

- 数据库应用领域

数据库的应用领域几乎涉及到了需要数据管理的方方面面，融机构、游戏网站、购物网站、论坛网站 ... 都需要数据库进行数据存储管理。

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-C9xUud7B-1610634103084)(./img/view.jpg)]

- 基本概念

- 数据库： 按照数据一定结构，存储管理数据的仓库。数据库是在数据库管理系统管理和控制下，在一定介质上的数据集合。
- 数据库管理系统： 管理数据库的软件，用于建立和维护数据库。
- 数据库系统： 由数据库和数据库管理系统，开发工具等组成的集合。

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-MmavvQwr-1610634103089)(img/数据库系统.png)]

- 数据库分类和常见数据库

- 关系型数据库和非关系型数据库

关系型： 采用关系模型（二维表）来组织数据结构的数据库

非关系型： 不采用关系模型组织数据结构的数据库

- 开源和非开源

开源： MySQL、SQLite、MongoDB

非开源： Oracle、DB2、SQL\_Server

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-9VTcoo9H-1610634103094)(img/databases.jpg)]

## 3.2 MySQL

1996年，MySQL 1.0发布,作者Monty Widenius, 为一个叫TcX的公司打工，当时只是内部发布。到了96年10月，MySQL 3.11.1发布了，一个月后，Linux版本出现了。真正的MySQL关系型数据库于1998年1月发行第一个版本。MySQL是个开源数据库，后来瑞典有了专门的MySQL开发公司，将该数据库发展壮大，在之后被Sun收购，Sun又被Oracle收购。

官网地址：<https://www.mysql.com/>

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-aFjEVR0K-1610634103100)(./img/mysql.jpg)]

- MySQL特点

1. 是开源数据库, 使用C和C++编写
2. 能够工作在众多不同的平台上
3. 提供了用于C、C++、Python、Java、Perl、PHP、Ruby众多语言的API
4. 存储结构优良, 运行速度快
5. 功能全面丰富

- MySQL数据库结构

数据元素 --> 记录 --> 数据表 --> 数据库

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-mqm0Z5gH-1610634103106)(img/库结构.png)]

- 基本概念解析

- 数据表 (table) : 存放数据的表格
- 字段 (column) : 每个列, 用来表示该列数据的含义
- 记录 (row) : 每个行, 表示一组完整的数据

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-xUnTB8Rq-1610634103110)(img/表结构.png)]

- MySQL安装

- Ubuntu安装MySQL服务

- 终端执行: `sudo apt install mysql-server`
    - 配置文件: `/etc/mysql`
    - 数据库存储目录: `/var/lib/mysql`

- Windows安装MySQL

- 下载MySQL安装包(windows)  
<https://dev.mysql.com/downloads/windows/installer/8.0.html>
    - 直接运行安装文件安装

- 启动和连接MySQL服务

- 服务端启动

- 查看MySQL状态: `sudo service mysql status`
    - 启动/停止/重启服务: `sudo service mysql start/stop/restart`

- 连接数据库

```
1 | mysql -h 主机地址 -u 用户名 -p
```

注意:

1. 回车后输入数据库密码 (我们设置的是123456)
2. 如果链接自己主机数据库可省略 -h 选项

- 关闭连接

```
1 | ctrl-D
2 | exit
```

### 3.3 SQL语言

- 什么是SQL

结构化查询语言(Structured Query Language)，一种特殊目的的编程语言，是一种数据库查询和程序设计语言，用于存取数据以及查询、更新和管理关系数据库系统。

- SQL语言特点
  - SQL语言基本上独立于数据库本身
  - 各种不同的数据库对SQL语言的支持与标准存在着细微的不同
  - 每条命令以 ; 结尾
  - SQL命令（除了数据库名和表名）关键字和字符串可以不区分字母大小写

### 3.4 数据库管理

#### 1. 查看已有库

```
show databases;
```

#### 2. 创建库

```
create database 库名 [character set utf8];
```

```
1 | e.g. 创建stu数据库，编码为utf8
2 | create database stu character set utf8;
3 | create database stu charset=utf8;
4 | create database stu default charset utf8;
```

注意：库名的命名

1. 数字、字母、下划线,但不能使用纯数字
2. 库名区分字母大小写
3. 不要使用特殊字符和mysql关键字

#### 3. 切换库

```
use 库名;
```

```
1 | e.g. 使用stu数据库
2 | use stu;
```

#### 4. 查看当前所在库

```
select database();
```

#### 5. 删除库

```
drop database 库名;
```

```
1 e.g. 删除test数据库
2 drop database test;
```

## 3.5 数据表管理

- 基本思考过程
  1. 确定存储内容
  2. 明确字段构成
  3. 确定字段数据类型

### 3.5.1 基础数据类型

- 数字类型：
  - 整数类型：INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT
  - 浮点类型：FLOAT, DOUBLE, DECIMAL
  - 比特值类型：BIT

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-VDJYYCEI-1610634103114)(img/整型.png)]

注意：

1. 对于准确性要求比较高的东西，比如money，用decimal类型减少存储误差。列的声明语法是DECIMAL(M,D)。M是数字的最大数字位数，D是小数点右侧数字的位数。比如DECIMAL(6,2)最多存6位数字，小数点后占2位,取值范围-9999.99到9999.99。
2. 比特值类型指0，1值表达2种情况，如真，假

- 字符串类型：
  - 普通字符串：CHAR, VARCHAR
  - 存储文本：TEXT
  - 存储二进制数据：BLOB
  - 存储选项型数据：ENUM, SET

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-dX9foTLD-1610634103117)(img/字符串.png)]

注意：

1. char：定长，即指定存储字节数后，无论实际存储了多少字节数据，最终都占指定的字节大小。默认只能存1字节数据。存取效率高。
2. varchar：不定长，效率偏低，但是节省空间，实际占用空间根据实际存储数据大小而定。必须要指定存储大小 varchar(50)
3. enum用来存储给出的多个值中的一个值,即单选，enum('A','B','C')
4. set用来存储给出的多个值中一个或多个值，即多选，set('A','B','C')

## 3.5.2 表的基本操作

- 1.创建表

```
create table 表名(字段名 数据类型,字段名 数据类型,...字段名 数据类型);
```

- 字段约束

- 如果你想设置数字为无符号则加上 unsigned
- 如果你不想字段为 NULL 可以设置字段的属性为 NOT NULL， 在操作数据库时如果输入该字段的数据为NULL， 就会报错。
- DEFAULT 表示设置一个字段的默认值
- AUTO\_INCREMENT定义列为自增的属性，一般用于主键，数值会自动加1。
- PRIMARY KEY 关键字用于定义列为主键。主键的值不能重复,且不能为空。

```
1 e.g. 创建班级表
2 create table class_1 (id int primary key auto_increment,name varchar(32) not
  null,age tinyint unsigned not null,sex enum('w','m'),score float default
  0.0);
3
4 e.g. 创建兴趣班表
5 create table interest (id int primary key auto_increment,name varchar(32) not
  null,hobby set('sing','dance','draw'),level char not null,price
  decimal(6,2),remark text);
```

- 2.查看数据表

```
show tables;
```

- 3.查看表结构

```
desc 表名;
```

- 4.删除表

```
drop table 表名;
```

## 3.6 表数据基本操作

### 3.5.1 插入(insert)

```
1 insert into 表名 values(值1),(值2),...;
2 insert into 表名(字段1,...) values(值1),...;
```

```
1 e.g.
2 insert into class_1 values (2,'Baron',10,'m',91),(3,'Jame',9,'m',90);
3
4 insert into class_1 (name,age,sex,score) values ('Lucy',17,'w',81);
5
```

### 3.6.2 查询(select)

```
1 select * from 表名 [where 条件];
2 select 字段1,字段2 from 表名 [where 条件];
```

```
1 e.g.
2 select * from class_1;
3 select name,age from class_1;
```

### 3.6.3 where子句

where子句在sql语句中扮演了重要角色，主要通过一定的运算条件进行数据的筛选，在查询，删除，修改中都有使用。

- 算数运算符

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-9WHqE2UZ-1610634103121)(img/算数.png)]

```
1 e.g.
2 select * from class_1 where age % 2 = 0;
```

- 比较运算符

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-84nXCAWE-1610634103126)(img/比较.png)]

```
1 e.g.
2 select * from class_1 where age > 8;
3 select * from class_1 where between 8 and 10;
4 select * from class_1 where age in (8,9);
```

- 逻辑运算符

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-XqUbasDU-1610634103131)(img/逻辑.png)]

```
1 e.g.
2 select * from class_1 where sex='m' and age>9;
```

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-jlAluwSh-1610634103135)(img/运算符.png)]

### 3.6.4 更新表记录(update)

```
1 update 表名 set 字段1=值1,字段2=值2,... where 条件;
2
3 注意:update语句后如果不加where条件,所有记录全部更新
```

```
1 e.g.
2 update class_1 set age=11 where name='Abby';
```

### 3.6.5 删除表记录 (delete)

```
1 delete from 表名 where 条件;
2
3 注意:delete语句后如果不加where条件,所有记录全部清空
```

```
1 e.g.
2 delete from class_1 where name='Abby';
```

### 3.6.6 表字段的操作(alter)

```
1 语法 : alter table 表名 执行动作;
2
3 * 添加字段(add)
4     alter table 表名 add 字段名 数据类型;
5     alter table 表名 add 字段名 数据类型 first;
6     alter table 表名 add 字段名 数据类型 after 字段名;
7 * 删除字段(drop)
8     alter table 表名 drop 字段名;
9 * 修改数据类型(modify)
10    alter table 表名 modify 字段名 新数据类型;
11 * 修改字段名(change)
12    alter table 表名 change 旧字段名 新字段名 新数据类型;
13 * 表重命名(rename)
14    alter table 表名 rename 新表名;
```

```
1 e.g.
2 alter table interest add tel char(11) after name;
```

### 3.5.7 时间类型数据

- 日期 : DATE
- 日期时间: DATETIME, TIMESTAMP
- 时间: TIME
- 年份 : YEAR

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-nmbzqOg4-1610634103141)(img/时间.png)]

- 时间格式

```
1 date : "YYYY-MM-DD"
2 time : "HH:MM:SS"
3 datetime : "YYYY-MM-DD HH:MM:SS"
4 timestamp : "YYYY-MM-DD HH:MM:SS"
```



注意:

1. datetime : 以系统时间存储
2. timestamp : 以标准时间存储但是查看时转换为系统时区, 所以表现形式和datetime 相同

```
1 e.g.  
2 create table marathon (id int primary key auto_increment,athlete  
   varchar(32),birthday date,registration_time datetime,performance time);
```

- 日期时间函数
  - now() 返回服务器当前日期时间,格式对应datetime类型
  - curdate() 返回当前日期, 格式对应date类型
  - curtime() 返回当前时间, 格式对应time类型
- 时间操作

时间类型数据可以进行比较和排序等操作, 在写时间字符串时尽量按照标准格式书写。

```
1 select * from marathon where birthday>='2000-01-01';  
2 select * from marathon where birthday>="2000-07-01" and  
   performance<="2:30:00";
```

## 3.7 高级查询语句

- 模糊查询和正则查询

### 1. 模糊查询

LIKE用于在where子句中进行模糊查询, SQL LIKE 子句中使用百分号%来表示任意0个或多个字符, 下划线\_表示任意一个字符。

```
1 SELECT field1, field2,...fieldN  
2 FROM table_name  
3 WHERE field1 LIKE condition1
```

```
1 e.g.  
2 mysql> select * from class_1 where name like 'A%';
```

### 2. 正则查询

mysql中对正则表达式的支持有限, 只支持部分正则元字符:

```
1 SELECT field1, field2,...fieldN  
2 FROM table_name  
3 WHERE field1 REGEXP condition1
```

```
1 e.g.  
2 select * from class_1 where name regexp '^B.+';
```

- as 用法

在sql语句中as用于给字段或者表重命名

```
1 | select name as 姓名,age as 年龄 from class_1;
2 | select * from class_1 as c where c.age > 17;
```

- 排序

ORDER BY 子句来设定你想按哪个字段哪种方式来进行排序，再返回搜索结果。

使用 ORDER BY 子句将查询数据排序后再返回数据：

```
1 | SELECT field1, field2,...fieldN from table_name1 where field1
2 | ORDER BY field1 [ASC [DESC]]
```

默认情况ASC表示升序，DESC表示降序

```
1 | select * from class_1 where sex='m' order by age desc;
```

复合排序：对多个字段排序，即当第一排序项相同时按照第二排序项排序

```
1 | select * from class_1 order by score desc,age;
```

- 限制

LIMIT 子句用于限制由 SELECT 语句返回的数据数量 或者 UPDATE,DELETE语句的操作数量

带有 LIMIT 子句的 SELECT 语句的基本语法如下：

```
1 | SELECT column1, column2, columnN
2 | FROM table_name
3 | WHERE field
4 | LIMIT [num]
```

- 联合查询

UNION 操作符用于连接两个以上的 SELECT 语句的结果组合到一个结果集合中。多个 SELECT 语句会删除重复的数据。

UNION 操作符语法格式：

```
1 | SELECT expression1, expression2, ... expression_n
2 | FROM tables
3 | [WHERE conditions]
4 | UNION [ALL | DISTINCT]
5 | SELECT expression1, expression2, ... expression_n
6 | FROM tables
7 | [WHERE conditions];
```

默认UNION后为 DISTINCT表示删除结果集中重复的数据。如果使用ALL则返回所有结果集，包含重复数据。

```
1 | select * from class_1 where sex='m' UNION ALL select * from class_1 where age
  | > 9;
```

- 子查询

- 定义： 当一个select语句中包含另一个select 查询语句，则称之为有子查询的语句
- 子查询出现的位置：
  - from 之后， 此时子查询的内容作为一个新的表内容， 再进行外层select查询

```
1 | select name from (select * from class_1 where sex='m') as s where  
  | s.score > 90;
```

注意： 需要将子查询结果集重命名一下， 方便where子句中的引用操作

- where字句中， 此时select查询到的内容作为外层查询的条件值

```
1 |      select * from class_1 where age = (select age from class_1 where  
  |      name='Tom');
```

注意： 子句结果作为一个值使用时， 返回的结果需要一个明确值， 不能是多行或者多列。 如果子句结果作为一个集合使用即where子句中是in操作则结果可以是一个字段的多个记录。

## 3.8 聚合操作

聚合操作指的是在数据查找基础上对数据的进一步整理筛选行为， 在认识聚合之前先看一个更完整的sql语句

- select语句执行顺序

```
1 | (7)      SELECT  
2 | (8)      [DISTINCT] <select_list>  
3 | (1)      FROM <left_table>  
4 | (3)      <join_type> JOIN <right_table>  
5 | (2)      ON <join_condition>  
6 | (4)      WHERE <where_condition>  
7 | (5)      GROUP BY <group_by_list>  
8 | (6)      HAVING <having_condition>  
9 | (9)      ORDER BY <order_by_condition>  
10 | (10)     LIMIT <limit_number>
```

### 3.8.1 聚合函数

方法	功能
avg(字段名)	该字段的平均值
max(字段名)	该字段的最大值
min(字段名)	该字段的最小值
sum(字段名)	该字段所有记录的和
count(字段名)	统计该字段记录的个数

eg1 : 找出表中的最大攻击力的值?

```
1 | select max(attack) from sanguo;
```

eg2 : 表中共有多少个英雄?

```
1 | select count(name) as number from sanguo;
```

eg3 : 蜀国英雄中攻击值大于200的英雄的数量

```
1 | select count(*) from sanguo where attack > 200;
```

注意：此时select 后只能写聚合函数，无法查找其他字段。

### 3.8.2 聚合分组

- group by

给查询的结果进行分组

e.g. : 计算每个国家的平均攻击力

```
1 | select country,avg(attack) from sanguo
2 | group by country;
```

e.g. : 对多个字段创建索引，此时多个字段都相同时为一组

```
1 | select age,sex,count(*) from class1 group by age,sex;
```

e.g. : 所有国家的男英雄中 英雄数量最多的前2名的 国家名称及英雄数量

```
1 | select country,count(id) as number from sanguo
2 | where gender='M' group by country
3 | order by number DESC
4 | limit 2;
```

注意：使用分组时select 后的字段为group by分组的字段和聚合函数，不能包含其他内容。  
group by也可以同时依照多个字段分组，如group by A, B 此时必须A,B两个字段值均相同才算一组。

### 3.8.3 聚合筛选

- having语句

对分组聚合后的结果进行进一步筛选

```
1 eg1 : 找出平均攻击力大于105的國家的前2名,显示國家名称和平均攻击力
2 select country,avg(attack) from sanguo
3 group by country
4 having avg(attack)>105
5 order by avg(attack) DESC
6 limit 2;
```

#### 注意

1. having语句必须与group by联合使用。
2. having语句存在弥补了where关键字不能与聚合函数联合使用的不足,where只能操作表中实际存在的字段。

### 3.8.4 去重语句

- distinct语句

不显示字段重复值

```
1 eg1 : 表中都有哪些國家
2 select distinct name,country from sanguo;
3 eg2 : 计算一共有多少个國家
4 select count(distinct country) from sanguo;
```

注意: distinct和from之间所有字段都相同才会去重

### 3.8.5 聚合运算

- 查询表记录时做数学运算

运算符: + - \* / %

```
1 eg1: 查询时显示攻击力翻倍
2 select name,attack*2 from sanguo;
3 eg2: 更新蜀国所有英雄攻击力 * 2
4 update sanguo set attack=attack*2 where country='蜀国';
```

## 3.9 索引操作

### 3.9.1 概述

- 定义

索引是对数据库表中一列或多列的值进行排序的一种结构,使用索引可快速访问数据库表中的特定信息。

- 优缺点

- 优点: 加快数据检索速度,提高查找效率
- 缺点: 占用数据库物理存储空间,当对表中数据更新时,索引需要动态维护,降低数据写入效率

- 适用性

适用情况: 数据量大,经常查询

### 3.9.2 索引分类

- 普通(MUL)

普通索引：字段值无约束,KEY标志为 MUL

- 唯一索引(UNI)

唯一索引(unique)：字段值不允许重复,但可为 NULL,KEY标志为 UNI

- 主键索引 (PRI)

一个表中只能有一个主键字段, 主键字段不允许重复,且不能为NULL, KEY标志为PRI。通常设置记录编号字段id,能唯一锁定一条记录

### 3.9.3 索引创建

- 创建表时直接创建索引

```
1 create table 表名(  
2  字段名 数据类型,  
3  字段名 数据类型,  
4  index 索引名(字段名), # 索引名可以省略  
5  index 索引名(字段名),  
6  unique 索引名(字段名)  
7 );
```

- 在已有表中创建索引:

```
1 create [unique] index 索引名 on 表名(字段名);
```

- 主键索引添加

```
1 alter table 表名 add primary key(id);
```

注意：

1. 通常我们只在经常进行查询操作的字段上创建索引
2. 对于数据量很少的表或者经常进行写操作而不是查询操作的表不适合创建索引

- 查看索引

```
1 1、desc 表名; --> KEY标志为: MUL 、 UNI。  
2 2、show index from 表名;
```

- 扩展：借助性能查看选项去查看索引性能

```

1 show variables like 'profiling';
2
3 set profiling = 1; 打开功能 （项目上线一般不打开）
4
5 show profiles 查看语句执行信息

```

- 删除索引

```

1 drop index 索引名 on 表名;
2 alter table 表名 drop primary key; # 删除主键

```

- 创建复合主键

```

1 primary key (uid,pid)
2 此时两个字段只要不都相同即可，用的不多。

```

## 3.10 外键约束和表关联关系

### 3.10.1 外键约束

- 约束：约束是一种限制，它通过对表的行或列的数据做出限制，来确保表的数据的完整性、唯一性
- foreign key 功能：建立表与表之间的某种约束的关系，由于这种关系的存在，能够让表与表之间的数据，更加的完整，关连性更强，为了具体说明创建如下部门表和人员表。
- 示例

```

1 # 创建部门表
2 CREATE TABLE dept (id int PRIMARY KEY auto_increment,dname VARCHAR(50) not null);

```

```

1 # 创建人员表
2 CREATE TABLE `person` (
3   `id` int PRIMARY KEY AUTO_INCREMENT,
4   `name` varchar(32) NOT NULL,
5   `age` tinyint DEFAULT 0,
6   `sex` enum('m','w','o') DEFAULT 'o',
7   `salary` decimal(8,2) DEFAULT 250.00,
8   `hire_date` date NOT NULL,
9   `dept_id` int
10  );

```

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-ipi0Q8rN-1610634103147)(img\部门-员工.jpg)]

上面两个表中每个人员都应该有指定的部门，但是实际上在没有约束的情况下人员是可以没有部门的或者也可以添加一个不存在的部门，这显然是不合理的。

- 主表和从表：若同一个数据库中，B表的外键与A表的主键相对应，则A表为主表，B表为从表。
- foreign key 外键的定义语法：

```

1  [CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
2  #给外键起名字 (名称)                                外键的索引名称
3  REFERENCES tbl_name (index_col_name, ...)
4
5  [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
6
7  [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]

```

该语法可以在 CREATE TABLE 和 ALTER TABLE 时使用

```

1  # 创建表时直接简历外键
2  CREATE TABLE `person` (
3      `id` int PRIMARY KEY AUTO_INCREMENT,
4      `name` varchar(32) NOT NULL,
5      `age` tinyint DEFAULT 0,
6      `sex` enum('m','w','o') DEFAULT 'o',
7      `salary` decimal(10,2) DEFAULT 250.00,
8      `hire_date` date NOT NULL,
9      `dept_id` int DEFAULT NULL,
10     constraint dept_fk foreign key(dept_id) references dept(id));

```

```

1  # 建立表后增加外键
2  alter table person add constraint dept_fk foreign key(dept_id) references
   dept(id);

```

注意：

1. 并不是任何情况表关系都需要建立外键来约束，如果没有类似上面的约束关系时也可以不 建立。
2. 从表的外键字段数据类型与指定的主表主键应该相同。

- 通过外键名称解除外键约束

```

1  alter table person drop foreign key dept_fk;

```

注意：删除外键后发现desc查看索引标志还在，其实外键也是一种索引，需要将外键名称的索引删除之后才可以。

```

1  show create table person

```

- 级联动作

- restrict(默认)：on delete restrict on update restrict
  - 当主表删除记录时，如果从表中有相关联记录则不允许主表删除
  - 当主表更改主键字段值时，如果从表有相关记录则不允许更改
- cascade：数据级联更新 on delete cascade on update cascade
  - 当主表删除记录或更改被参照字段的值时,从表会级联更新
- set null：on delete set null on update set null
  - 当主表删除记录时，从表外键字段值变为null



- 当主表更改主键字段值时，从表外键字段值变为null

### 3.10.2 表关联设计

当我们应对复杂的数据关系的时候，数据表的设计就显得尤为重要，认识数据之间的依赖关系是更加合理创建数据表关联性的前提。常见的数据关系如下：

- 一对一关系

一张表的一条记录一定只能与另外一张表的一条记录进行对应，反之亦然。

举例：学生信息和学籍档案，一个学生对应一个档案，一个档案也只属于一个学生

```
1 create table student(id int primary key auto_increment,name varchar(50) not
  null);
2
3 create table record(id int primary key auto_increment,
4 comment text not null,
5 st_id int unique,
6 foreign key(st_id) references student(id) # 约束
7 on delete cascade
8 on update cascade
9 );
```

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-nfSRm5l8-1610634103150)(img/一对一关系.jpg)]

- 一对多关系

一张表中有一条记录可以对应另外一张表中的多条记录；但是反过来，另外一张表的一条记录只能对应第一张表的一条记录，这种关系就是一对多或多对一

举例：一个人可以拥有多辆汽车，每辆车登记的车主只有一人。

```
1 create table person(
2   id varchar(32) primary key,
3   name varchar(30),
4   sex char(1),
5   age int
6 );
7
8 create table car(
9   id varchar(32) primary key,
10  name varchar(30),
11  price decimal(10,2),
12  pid varchar(32),
13  constraint car_fk foreign key(pid) references person(id)
14 );
```

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-gC5LLLLM8-1610634103154)(img\一对多.jpg)]

- 多对多关系

一对表中（A）的一条记录能够对应另外一张表（B）中的多条记录；同时B表中的一条记录也能对应A表中的多条记录

举例：一个运动员可以报多个项目，每个项目也会有多个运动员参加,这时为了表达多对多关系需要单独创建关系表。

```
1 CREATE TABLE `athlete` (  
2   `id` int NOT NULL AUTO_INCREMENT,  
3   `name` varchar(30),  
4   `age` tinyint NOT NULL,  
5   `country` varchar(30) NOT NULL,  
6   `description` varchar(30),  
7   PRIMARY KEY (`id`)  
8 );  
9  
10 CREATE TABLE `item` (  
11   `id` int NOT NULL AUTO_INCREMENT,  
12   `rname` varchar(30) NOT NULL,  
13   PRIMARY KEY (`id`)  
14 );  
15  
16 CREATE TABLE `athlete_item` (  
17   `aid` int NOT NULL,  
18   `tid` int NOT NULL,  
19   PRIMARY KEY (`aid`, `tid`),  
20   CONSTRAINT `athlete_fk` FOREIGN KEY (`aid`) REFERENCES `athlete` (`id`),  
21   CONSTRAINT `item_fk` FOREIGN KEY (`tid`) REFERENCES `item` (`id`)  
22 );
```

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-F1nsP0PP-1610634103165)(img\多对多.jpg)]

### 3.10.3 E-R模型

- 定义

- 1 E-R模型(Entry-Relationship)即 实体-关系 数据模型,用于数据库设计
- 2 用简单的图(E-R图)反映了现实世界中存在的事物或数据以及他们之间的关系

- 实体、属性、关系

#### 实体

- 1 1、描述客观事物的概念
- 2 2、表示方法：矩形框
- 3 3、示例：一个人、一本书、一杯咖啡、一个学生

#### 属性

- 1 1、实体具有的某种特性
- 2 2、表示方法：椭圆形
- 3 3、示例
- 4 学生属性：学号、姓名、年龄、性别、专业 ...
- 5 感受属性：悲伤、喜悦、刺激、愤怒 ...

#### 关系

- 1 1、实体之间的联系
- 2 2、一对一关联(1:1)
- 3 3、一对多关联(1:n)
- 4 4、多对多关联(m:n)

- ER图的绘制

矩形框代表实体,菱形框代表关系,椭圆形代表属性

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-OMbk1pf3-1610634103167)(./img/er.PNG)]

### 3.10.4 表关联整理

如果多个表存在一定关联关系,可以多表在一起进行查询操作,其实表的关联整理与外键约束之间并没有必然联系,但是基于外键约束设计的具有关联性的表往往会更多使用关联查询查找数据。

- 简单多表查询

多个表数据可以联合查询,语法格式如下:

```
1 select  字段1,字段2... from 表1,表2... [where 条件]
```

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-gnTUT4IP-1610634103173)(img\QQ截图20200331214216.jpg)]

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-XUPsRpd3-1610634103175)(img\QQ截图20200331214259.jpg)]

```
1 e.g.
2 select * from dept, person where dept.id = person.dept_id;
```

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-JJCdTVER-1610634103179)(img\QQ截图20200331213254.jpg)]

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-WBEI681i-1610634103182)(img\QQ截图20200331213413.jpg)]

- 内连接

内连接查询只会查找到符合条件的记录,其实结果和表关联查询是一样的,官方更推荐使用内连接查询。

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-TmPwSpNf-1610634103186)(./img/inner.PNG)]

```
1 SELECT  字段列表
2 FROM 表1 INNER JOIN 表2
3 ON 表1.字段 = 表2.字段;
```

```
1 select * from person inner join dept on person.dept_id =dept.id;
```

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-wcKvBOc0-1610634103191)(img\QQ截图20200331213918.jpg)]

- 笛卡尔积

笛卡尔积就是将A表的每一条记录与B表的每一条记录强行拼在一起。所以，如果A表有n条记录，B表有m条记录，笛卡尔积产生的结果就会产生n\*m条记录。

```
1 | select * from person inner join dept;
```

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-WLo2GeAl-1610634103196)(img\QQ截图20200331214528.jpg)]

- 左连接：左表为主表，显示右表中与左表匹配的项

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-36IM5oyE-1610634103199)(./img/left.PNG)]

```
1 | SELECT 字段列表
2 |     FROM 表1 LEFT JOIN 表2
3 | ON 表1.字段 = 表2.字段;
```

```
1 | select * from person left join dept on person.dept_id =dept.id;
```

- 右连接：右表为主表，显示左表中与左表匹配的项

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-1b4rGVxM-1610634103201)(./img/right.PNG)]

```
1 | SELECT 字段列表
2 |     FROM 表1 RIGHT JOIN 表2
3 | ON 表1.字段 = 表2.字段;
```

```
1 | select * from person right join dept on person.dept_id =dept.id;
```

注意：我们尽量使用数据量大的表作为基准表，即左连接做左表，右连接做右表

## 3.11 视图

- 视图概念

视图是存储的查询语句,当调用的时候,产生结果集,视图充当的是虚拟表的角色。其实视图可以理解为一个表或多个表中导出出来的表，作用和真实表一样，包含一系列带有行和列的数据 视图中，用户可以使用SELECT语句查询数据，也可以使用INSERT，UPDATE，DELETE修改记录，视图可以使用户操作方便，并保障数据库系统安全，如果原表改名或者删除则视图也失效。

- 创建视图

```
1  语法结构：
2
3  CREATE [OR REPLACE] VIEW [view_name] AS [SELECT_STATEMENT];
4
5  释义：
6
7  CREATE VIEW: 创建视图
8  OR REPLACE : 可选，如果添加原来有同名视图的情况下会覆盖掉原有视图
9  view_name : 视图名称
10 SELECT_STATEMENT : SELECT语句
11
12 e.g.
13 create view c1 as select name,age from class_1;
```

- 视图表的增删改查操作

视图的增删改查操作与一般表的操作相同，使用insert update delete select即可，但是原数据表的约束条件仍然对视图产生作用。

- 查看现有视图

```
1 show full tables in stu where table_type like 'VIEW';
```

- 删除视图

drop view [IF EXISTS] 视图名;

IF EXISTS 表示如果存在，这样即使没有指定视图也不会报错。

```
1 drop view if exists c1;
```

- 修改视图

参考创建视图，将create关键字改为alter

```
1 alter view c1 as select name,age,score from class_1;
```

- 视图作用

- 作用

1. 是对数据的一种重构，不影响原数据表的使用。
2. 简化高频复杂操作的过程，就像一种对复杂操作的封装。
3. 提高安全性，可以给不同用户提供不同的视图。
4. 让数据更加清晰。

- 缺点

1. 视图的性能相对较差，从数据库视图查询数据可能会很慢。

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-ulwANPsX-1610634103207)(img\QQ截图20200331224854.jpg)]

## 3.12 函数和存储过程

存储过程和函数是事先经过编译并存储在数据库中的一段sql语句集合，调用存储过程和函数可以简化应用开发工作，提高数据处理的效率。

### 3.12.1 函数创建

```
1  delimiter 自定义符号    -- 如果函数体只有一条语句，begin和end可以省略，同时delimiter
   也可以省略
2
3      create function 函数名(形参列表) returns 返回类型    -- 注意是retruns
4
5      begin
6
7          函数体        -- 函数语句集,set @a 定义变量
8
9          return val
10
11     end 自定义符号
12
13 delimiter ;
14
15 释义：
16 delimiter 自定义符号 是为了在函数内些语句方便，制定除了;之外的符号作为函数书写结束标志，一
   般用$$或者//
17 形参列表： 形参名 类型    类型为mysql支持类型
18 返回类型： 函数返回的数据类型,mysql支持类型即可
19 函数体： 若干sql语句组成，如果只有一条语句也可以不写delimiter和begin,end
20 return: 返回指定类型返回值
```

```
1  e.g. 无参数的函数调用
2  delimiter $$
3  create function st() returns int
4  begin
5  return (select score from class_1 order by score desc limit 1);
6  end $$
7  delimiter ;
8
9  select st();
```

```
1  e.g. 含有参数的函数调用
2  delimiter $$
3  create function queryNameById(uid int(10))
4  returns varchar(20)
5  begin
6  return (select name from class_1 where id=uid);
7  end $$
8  delimiter ;
9
10 select queryNameById(1);
```

- 设置变量
  - 用户变量方法： set @[变量名] = 值; 使用时用@[变量名]。

- 局部变量：在函数内部设置 `declare [变量名] [变量类型]`；局部变量可以使用`set`赋值或者使用`into`关键字。

```

1  delimiter $$
2  create function st2()
3  returns int
4  begin
5  declare val1 float;
6  declare val2 float;
7  set val1=(select score from cls order by score desc limit 1);
8  select score from cls order by score limit 1 into val2;
9  return val1+val2;
10 end $$

```

### 3.12.2 存储过程创建

创建存储过程语法与创建函数基本相同，但是没有返回值。

```

1  delimiter 自定义符号
2
3      create procedure 存储过程名(形参列表)
4
5      begin
6
7          存储过程          -- 存储过程语句集,set @a 定义变量
8
9      end 自定义符号

```

```

11 delimiter ;

```

释义：

`delimiter` 自定义符号 是为了在函数内些语句方便，制定除了`;`之外的符号作为函数书写结束标志

形参列表：[ `IN` | `OUT` | `INOUT` ] 形参名 类型

`in` 输入, `out` 输出, `inout` 可以输入也可以输出

存储过程：若干`sql`语句组成，如果只有一条语句也可以不写`delimiter`和`begin,end`

```

1  e.g. 存储过程创建和调用
2  delimiter $$
3  create procedure st()
4  begin
5      select name,age from class_1;
6      select name,score from class_1 order by score desc;
7  end $$
8  delimiter ;
9
10 call st();

```

- 存储过程三个参数的区别
  - `IN` 类型参数可以接收变量也可以接收常量，传入的参数在存储过程内部使用即可，但是在存储过程内部的修改无法传递到外部。

- OUT 类型参数只能接收一个变量，接收的变量不能够在存储过程内部使用（内部为NULL），但是可以在存储过程内对这个变量进行修改。因为定义的变量是全局的，所以外部可以获取这个修改后的值。
- INOUT类型参数同样只能接收一个变量，但是这个变量可以在存储过程内部使用。在存储过程内部的修改也会传递到外部。

```

1  e.g. : 分别将参数类型改为IN OUT INOUT 看一下结果区别
2  delimiter $$
3  create procedure p_out ( OUT num int )
4  begin
5      select num;
6      set num=100;
7      select num;
8  end $$
9
10 delimiter ;
11
12 set @num=10;
13 call p_out(@num)

```

### 3.12.3 存储过程和存储函数操作

#### 1. 调用存储过程

语法：

```
1 call 存储过程名字 ([存储过程的参数[,.....]])
```

#### 2. 调用存储函数

语法：

```
1 select 存储函数名字 ([存储过程的参数[,.....]])
```

#### 3. 使用show status语句查看存储过程和函数的信息

语法：

```
1 show {procedure|function} status [like'存储过程或存储函数的名称']
```

显示内容：数据库、名字、类型、创建者、创建和修改日期

#### 4. 使用show create语句查看存储过程和函数的定义

语法：

```
1 show create {procedure|function} 存储过程或存储函数的名称
```

#### 5. 查看所有函数或者存储过程



```
1 | select name from mysql.proc where db='stu' and  
   | type='[procedure/function]';
```

## 6. 删除存储过程或存储函数

语法:

```
1 | DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

### 3.12.4 函数和存储过程区别

1. 函数有且只有一个返回值，而存储过程不能有返回值。
2. 函数只能有输入参数，而存储过程可以有in,out,inout多个类型参数。
3. 存储过程中的语句功能更丰富，实现更复杂的业务逻辑，可以理解为一个按照预定步骤调用的执行过程，而函数中不能展示查询结果集语句，只是完成查询的工作后返回一个结果，功能针对性比较强。
4. 存储过程一般是作为一个独立的部分来执行(call调用)。而函数可以作为查询语句的一个部分来调用。

## 3.13 事务控制

### 3.13.1 事务概述

MySQL 事务主要用于处理操作量大，复杂度高的数据。比如说，在人员管理系统中，你删除一个人员，既需要删除人员的基本资料，也要删除和该人员相关的信息，如信箱，文章等等，如果操作就必须同时操作成功，如果有一个不成功则所有数据都不动。这时候数据库操作语句就构成一个事务。事务主要处理数据的增删改操作。

- 定义

一件事从开始发生到结束的过程

- 作用

确保数据操作过程中的安全。

### 3.13.2 事务操作

#### 1. 开启事务

```
1 | mysql>begin; # 方法1
```

2. 开始执行事务中的若干条SQL命令（增删改）
3. 终止事务，若begin之后使用commit提交事务或者使用rollback进行事务回滚。

```
1 | mysql>commit; # 事务中SQL命令都执行成功,提交到数据库,结束!  
2 | mysql>rollback; # 有SQL命令执行失败,回滚到初始状态,结束!
```

### 3.13.3 事务四大特性

#### 1. 原子性 (atomicity)

一个事务必须视为一个不可分割的最小工作单元，对于一个事务来说，不可能只执行其中的一部分操作,整个事务中的所有操作要么全部提交成功，要么全部失败回滚

#### 2. 一致性 (consistency)

事务完成时，数据必须处于一致状态，数据的完整性约束没有被破坏。

#### 3. 隔离性 (isolation)

数据库允许多个并发事务同时对其数据进行读写和修改的能力，而多个事务相互独立。隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。

#### 4. 持久性 (durability)

一旦事务提交，则其所做的修改就会永久保存到数据库中。此时即使系统崩溃，修改的数据也不会丢失。

### 3.13.4 事务隔离级别

事务四大特性中的隔离性是在使用事务时最为需要注意的特性，因为隔离级别不同带来的操作现象也有区别

- 隔离级别

- 读未提交：read uncommitted

事物A和事物B，事物A未提交的数据，事物B可以读取到  
这里读取到的数据叫做“脏数据”（不确定的）  
这种隔离级别最低，这种级别一般是在理论上存在，数据库隔离级别一般都高于该级别

- 读已提交：read committed

事物A和事物B，事物A提交的数据，事物B才能读取到  
这种隔离级别高于读未提交  
换句话说，对方事物提交之后的数据，我当前事物才能读取到  
这种级别可以避免“脏数据”  
这种隔离级别会导致“不可重复读取”

- 可重复读：repeatable read

事务A和事务B，事务A提交之后的数据，事务B读取不到  
事务B是可重复读取数据  
这种隔离级别高于读已提交  
MySQL默认级别  
虽然可以达到可重复读取，但是会导致“幻像读”

- 串行化：serializable

事务A和事务B，事务A在操作数据库时，事务B只能排队等待  
这种隔离级别很少使用，吞吐量太低，用户体验差  
这种级别可以避免“幻像读”，每一次读取的都是数据库中真实存在数据，事务A与事务B  
串行，而不并发

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-hFJvklkv-1610634103211)(./img/隔离.png)]

## 3.14 数据库优化

### 3.14.1 数据库设计范式

设计关系数据库时，遵从不同的规范要求，设计出合理的关系型数据库，这些不同的规范要求被称为不同的范式。

目前关系数据库有六种范式：第一范式（1NF）、第二范式（2NF）、第三范式（3NF）、巴斯-科德范式（BCNF）、第四范式（4NF）和第五范式（5NF，又称完美范式）。

各种范式呈递次规范，越高的范式数据库冗余越小。但是范式越高也意味着表的划分更细，一个数据库中需要的表也就越多，此时多个表联接在一起的花费是巨大的，尤其是当需要连接的两张或者多张表数据非常庞大的时候，表连接操作几乎是一个噩梦，这严重地降低了系统运行性能。所以通常数据库设计遵循第一第二第三范式，以避免数据操作异常，又不至于表关系过于复杂。

范式简介：

- 第一范式：数据库表的每一列都是不可分割的原子数据项，而不能是集合，数组，记录等组合的数据项。简单来说要求数据库中的表示二维表，每个数据元素不可再分。  
例如：在国内的话通常理解都是姓名是一个不可再拆分的单位，这时候就符合第一范式；但是在国外的话还要分为FIRST NAME和LAST NAME，这时候姓名这个字段就是还可以拆分为更小的单位的字段，就不符合第一范式了。
- 第二范式：第二范式（2NF）要求数据库表中的每个实例或记录必须可以被唯一地区分，所有属性依赖于主属性。即选取一个能区分每个实体的属性或属性组，作为实体的唯一标识，每个属性都能被主属性筛选。其实简单理解要设置一个区分各个记录的主键就好了。
- 第三范式：在第二范式的基础上属性不传递依赖，即每个属性不依赖其他非主属性。要求一个表中不包含已在其它表中包含的非主关键字信息。其实简单来说就是合理使用外键，使不同的表中不要有重复的字段就好了。

### 3.14.2 MySQL存储引擎

- **定义：**mysql数据库管理系统中用来处理表的处理器（操作数据库系统的核心代码）
- **基本操作**

```
1 1、查看所有存储引擎
2  mysql> show engines;
3 2、查看已有表的存储引擎
4  mysql> show create table 表名;
5 3、创建表指定
6  create table 表名(...)engine=MyISAM,charset=utf8,auto_increment=10000;
7 4、已有表指定
8  alter table 表名 engine=InnoDB; 也可以更改自增值
```

- **常用存储引擎特点**

InnoDB

- 1 1、支持行级锁,仅对指定的记录进行加锁,这样其它进程还是可以对同一个表中的其它记录进行操作。  
(不允许两个客户端操作同一行数据)
- 2 2、支持外键、事务、事务回滚
- 3 3、表字段和索引同存储在一个文件中
- 4 1、表名 `.frm` : 表结构
- 5 2、表名 `.ibd` : 表记录及索引文件

## MyISAM

- 1 1、支持表级锁,在锁定期间,其它进程无法对该表进行写操作。如果你是写锁,则其它进程则读也不允许
- 2 2、表字段和索引分开存储
- 3 1、表名 `.frm` : 表结构
- 4 2、表名 `.MYI` : 索引文件(my index)
- 5 3、表名 `.MYD` : 表记录(my data)
- 6 写操作效率较低

### • 如何选择存储引擎

- 1 1、执行查操作多的表用 `MyISAM`(使用InnoDB浪费资源)
- 2 2、执行写操作多的表用 `InnoDB`
- 3
- 4 `CREATE TABLE tb_stu(  
5 `id` int(11) NOT NULL AUTO_INCREMENT,  
6 `name` varchar(30) DEFAULT NULL,  
7 `sex` varchar(2) DEFAULT NULL,  
8 PRIMARY KEY (`id`)  
9 )ENGINE=MyISAM DEFAULT CHARSET=utf8;  
10`

### 3.14.3 字段数据类型选择

- 优先程度 数字 > 时间日期 > 字符串
- 同一级别 占用空间小的 > 占用空间多的

- 1 字符串在查询比较排序时数据处理慢
- 2 占用空间少,数据库占磁盘页少,IO处理就更快

- 对数据存储精确不要求 float > decimal
- 如果很少被查询可以用 `TIMESTAMP` (时间戳实际是整形存储)

### 3.14.4 键的设置

- InnoDB如果不设置主键也会自己设置隐含的主键,所以最好自己设置
- 尽量设置占用空间小的字段为主键
- 外键的设置用于保持数据完整性,但是会降低数据导入和操作效率,特别是高并发情况下,而且会增加维护成本
- 虽然高并发下不建议使用外键约束,但是在表关联时建议在关联键上建立索引,以提高查找速度

### 3.14.5 explain语句

使用 EXPLAIN 关键字可以模拟优化器执行SQL查询语句，从而知道MySQL是如何处理你的SQL语句的。这可以帮你分析你的查询语句或是表结构的性能瓶颈。通过explain命令可以得到：

- 表的读取顺序
- 数据读取操作的操作类型
- 哪些索引可以使用
- 哪些索引被实际使用
- 表之间的引用
- 每张表有多少行被优化器查询

```
1 | explain select * from class_1 where id <5;
```

EXPLAIN主要字段解析：

- table：显示这一行的数据是关于哪张表的
- type：这是最重要的字段之一，显示查询使用了何种类型。从最好到最差的连接类型为system、const、eq\_reg、ref、range、index和ALL，一般来说，得保证查询至少达到range级别，最好能达到ref。

```
1 | type中包含的值：
2 | - system、const： 可以将查询的变量转为常量。如id=1；id为 主键或唯一键。
3 | - eq_ref： 访问索引，返回某单一行的数据。（通常在联接时出现，查询使用的索引为主键或唯一键）
4 | - ref： 访问索引，返回某个值的数据。（可以返回多行）通常使用=时发生
5 | - range： 这个连接类型使用索引返回一个范围中的行，比如使用>或<查找东西，并且该字段上建有索引时发生的情况
6 | - index： 以索引的顺序进行全表扫描，优点是不用排序，缺点是还要全表扫描
7 | - ALL： 全表扫描，应该尽量避免
```

- possible\_keys：显示可能应用在这张表中的索引。如果为空，表示没有可能应用的索引。
- key：实际使用的索引。如果为NULL，则没有使用索引。
- key\_len：使用的索引的长度。在不损失精确性的情况下，长度越短越好
- rows：MySQL认为必须检索的用来返回请求数据的行数

### 3.14.6 SQL优化

- 尽量选择数据类型占空间少，在where，group by，order by中出现的频率高的字段建立索引
- 尽量避免使用 select \* ...;用具体字段代替\*，不要返回用不到的任何字段
- 少使用like %查询，否则会全表扫描
- 控制使用自定义函数
- 单条查询最后添加 LIMIT 1，停止全表扫描
- where子句中不使用 !=，否则放弃索引全表扫描
- 尽量避免 NULL 值判断，否则放弃索引全表扫描  
优化前：select number from t1 where number is null;  
优化后：select number from t1 where number=0;
  - 在number列上设置默认值0，确保number列无NULL值

- 尽量避免 or 连接条件,否则会放弃索引进行全表扫描, 可以用union代替  
优化前: select id from t1 where id=10 or id=20;  
优化后: select id from t1 where id=10 union all  
select id from t1 where id=20;
- 尽量避免使用 in 和 not in,否则会全表扫描  
优化前: select id from t1 where id in(1,2,3,4);  
优化后: select id from t1 where id between 1 and 4;

### 3.14.7 表的拆分

垂直拆分: 表中列太多, 分为多个表, 每个表是其中的几个列。将常查询的放到一起, blob或者text类型字段放到另一个表

水平拆分: 减少每个表的数据量, 通过关键字进行划分然后拆成多个表

## 3.15 数据库备份和用户管理

### 3.15.1 表的复制

- 1、表能根据实际需求复制数据
- 2、复制表时不会把KEY属性复制过来

语法

```
1 | create table 表名 select 查询命令;
```

### 3.15.2 数据备份

1. 备份命令格式

```
mysqldump -u用户名 -p 源库名 > ~/stu.sql
```

2. 恢复命令格式

```
mysql -u root -p 目标库名 < stu.sql
```

目标库名必须存在

### 3.15.3 用户权限管理

开启MySQL远程连接

```

1  更改配置文件，重启服务！
2  1.cd /etc/mysql/mysql.conf.d
3  2.sudo vi mysqld.cnf  找到44行左右，加 # 注释
4      # bind-address = 127.0.0.1 #本机测试地址
5
6  3.保存退出
7  4.sudo service mysql restart
8  5.进入mysql修改用户表host值
9      use mysql;
10     update user set host='%' where user='root';
11 6.刷新权限
12     flush privileges;

```

## 添加授权用户

```

1  1. 用root用户登录mysql
2      mysql -u root -p
3  2. 添加用户 % 表示自动选择可用IP
4      CREATE USER 'username'@'host' IDENTIFIED BY 'password';
5  create user 'zmj'@'%' identified by 'zmj@2020';
6  3. 授权
7      grant 权限列表 on 库.表 to "用户名"@"%" [identified by "密码"] with grant
      option;
8  grant select,insert,update,delete,create,drop,create user on *.* to
      'zmj'@'%' with grant option;
9      revoke insert,update,select on 库.表 from 'user'@'%';
10     revoke create,drop on *.* from 'zmj'@'%';
11 4. 刷新权限
12     flush privileges;
13 5. 删除用户
14     drop user "用户名"@"%"

```

## 查看用户

```

1  1. use mysql;
2  2.select user from user;

```

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-kWKdCyny-1610634103215)(img\QQ截图20200401233542.jpg)]

## 权限列表

```

1  all privileges 、select 、insert , update, delete, alter,create,drop,create
   user等。
2  库.表 : *.* 代表所有库的所有表

```

## 示例

```

1  1. 创建用户
2      mysql>create user 'work'@'%' identified by '123';
3  2. 添加授权用户work,密码123,对所有库的所有表有所有权限
4      mysql>grant all privileges on *.* to 'work'@'%' identified by '123' with
grant option;
5      mysql>flush privileges;
6  3. 添加用户duty, 密码123,对db2库中所有表有所有权限
7      mysql>grant all privileges on books.* to 'duty'@'%' identified by '123'
with grant option;
8      mysql>flush privileges;

```

## 3.16 pymysql模块

pymysql是一个第三方库，如果自己的计算机上没有可以在终端使用命令进行安装。

```
1  sudo pip3 install pymysql
```

- pymysql使用流程

1. 建立数据库连接(db = pymysql.connect(...))
2. 创建游标对象(cur = db.cursor())
3. 游标方法: cur.execute("insert ....")
4. 提交到数据库或者获取数据 : db.commit()/db.fetchall()
5. 关闭游标对象 : cur.close()
6. 断开数据库连接 : db.close()

注：3, 4核心步骤

- 常用函数

```

1  db = pymysql.connect(参数列表)
2  功能：链接数据库
3
4  host : 主机地址,本地 localhost
5  port : 端口号,默认3306
6  user : 用户名
7  password : 密码
8  database : 库
9  charset : 编码方式,推荐使用 utf8
10 注: host,port 网络链接
11
12 db = pymysql.connect(host="localhost",
13                       port=3306,
14                       user="root",
15                       password="123456",
16                       database="stu",
17                       charset="utf8")

```



```
1 cur = db.cursor()
2 功能: 创建游标
3 返回值: 返回游标对象,用于执行具体SQL命令
```

```
1 cur.execute(sql,list_)
2 功能: 执行SQL命令
3 参数: sql sql语句
4 list_ 列表,用于给sql语句传递参量
5 """
6     Execute a query
7
8     :param str query: Query to execute.
9
10    :param args: parameters used with query. (optional)
11    :type args: tuple, list or dict
12
13    :return: Number of affected rows
14    :rtype: int
15
16    If args is a list or tuple, %s can be used as a placeholder in the
17    query.
18    If args is a dict, %(name)s can be used as a placeholder in the
19    query.
20    """
21 cur.executemany(sql命令,list_)
22 功能: 多次执行SQL命令,执行次数由列表中元组数量决定
23 参数: sql sql语句
24 list_ 列表中包含元组 每个元组用于给sql语句传递参量,一般用于写操作。
25 :return: Number of rows affected, if any.
```

```
1 # 查询数据
2 sql = "select name,age,sex,score from cls where score > 80;"
3 cur.execute(sql)
4
5 # 第一种
6 # for i in cur:
7 #     print(i) # 一条查询记录
8
9 # 第二种
10 # one = cur.fetchone()
11 # print(one)
12 # many = cur.fetchmany(2)
13 # print(many)
14 all = cur.fetchall()
15 print(all)
16
17 第三种
18 for i in range(cur.execute(sql)):
19     one = cur.fetchone()
20     print(one)
21
```

```
22 cur.fetchone() 获取查询结果集的第一条数据，查找到返回一个元组否则返回None """Fetch
the next row"""
23 cur.fetchmany(n) 获取前n条查找到的记录，返回结果为元组嵌套元组， ((记录1),(记录2))，查
询不到内容返回空元组。
24 cur.fetchall() 获取所有查找到的记录，返回结果形式同上。
25 cur.close() 关闭游标对象
```

```
1 try:
2     cur.executemany(sql,args_list)
3     db.commit()
4 except:
5     db.rollback()
6
7 db.commit() 提交到数据库执行
8 db.rollback() 回滚，用于当commit()出错是回复到原来的数据形态
9 db.close() 关闭连接
```

- 文件存储
  - 存储文件路径
    - 优点：节省数据库空间，提取方便
    - 缺点：文件或者数据库发生迁移会导致文件丢失
  - 存储文件本身
    - 优点：安全可靠，数据库在文件就在
    - 缺点：占用数据库空间大，文件存取效率低

