

公告

wiki和教程：[www.pythonav.com](http://www.pythonav.com)

免费教学视频：[B站](#)：凸头统治地球

高级专题教程：[网易云课堂](#)：武沛齐

聊技术，加武Sir微信



武沛齐

扫一扫上面的二维码图案，加我微信



Python技术交流群：737658057

软件测试开发交流群：721023555

昵称：武沛齐  
园龄：8年1个月  
粉丝：9974  
关注：44  
[+加关注](#)

我的标签

- Python(17)
- ASP.NET MVC(15)
- python之路(7)
- Tornado源码分析(5)
- 每天一道Python面试题(5)
- crm项目(4)
- 面试都在问什么？(2)
- Python开源组件 - Tyrion(1)
- Python面试315题(1)
- Python企业面试题讲解(1)

积分与排名

积分 - 426642  
排名 - 785

随笔分类


- JavaScript(1)
- MVC(15)
- Python(17)
- 面试都在问什么系列？【图】(2)
- 其他(37)

随笔 - 140 文章 - 164 评论 - 893


Python开发【第十八篇】：MySQL（二）

视图

视图是一个虚拟表（非真实存在），其本质是【根据SQL语句获取动态的数据集，并为其命名】，用户使用时只需使用【名称】即可获取结果集，并可以将其当作表来使用。



```
SELECT
*
FROM
(
    SELECT
        nid,
        NAME
    FROM
        tbl
    WHERE
        nid > 2
) AS A
WHERE
A. NAME > 'alex';
```




1、创建视图



```
--格式：CREATE VIEW 视图名称 AS SQL语句
CREATE VIEW v1 AS
SELET nid,
    name
FROM
    A
WHERE
    nid > 4
```



2、删除视图



```
--格式：DROP VIEW 视图名称
DROP VIEW v1
```

3、修改视图

View Code

4、使用视图

使用视图时，将其当作表进行操作即可，由于视图是虚拟表，所以无法使用其对真实表进行创建、更新和删除操作，仅能做查询用。



```
select * from v1
```

企业面试题及答案(1)  
请求响应(6)  
设计模式(9)  
微软C#(34)

随笔档案

- 2020年6月(1)
- 2020年5月(1)
- 2019年11月(1)
- 2019年10月(1)
- 2019年9月(4)
- 2018年12月(1)
- 2018年8月(1)
- 2018年5月(2)
- 2018年4月(1)
- 2017年8月(1)
- 2017年5月(1)
- 2017年3月(1)
- 2016年10月(1)
- 2016年7月(1)
- 2015年10月(1)
- 2015年8月(1)
- 2015年7月(1)
- 2015年6月(2)
- 2015年4月(2)
- 2014年3月(3)
- 2014年1月(3)
- 2013年12月(2)
- 2013年11月(2)
- 2013年10月(7)
- 2013年8月(17)
- 2013年7月(1)
- 2013年6月(14)
- 2013年5月(23)
- 2013年4月(3)
- 2013年3月(13)
- 2013年2月(1)
- 2012年11月(26)

相册

git(14)

最新评论

1. Re:Python开发【第十九篇】：Python  
操作MySQL  
执行本网页中第一段SQL语句，其中执行语  
句改成： cursor.execute('insert into  
part(caption) values("AB") ') 报错  
unable to reso...  
--serene1979

2. Re:python 面向对象（进阶篇）  
class Foo:

pass

这个类不是由,object.new(cls)创建的吗

触发器

对某个表进行【增/删/改】操作的前后如果希望触发某个特定的行为时，可以使用触发器，触发器用于定制用户对表的行进行【增/删/改】前后的行为。

1、创建基本语法

```
# 插入前
CREATE TRIGGER tri_before_insert_tb1 BEFORE INSERT ON tb1 FOR EACH ROW
BEGIN
    ...
END

# 插入后
CREATE TRIGGER tri_after_insert_tb1 AFTER INSERT ON tb1 FOR EACH ROW
BEGIN
    ...
END

# 删除前
CREATE TRIGGER tri_before_delete_tb1 BEFORE DELETE ON tb1 FOR EACH ROW
BEGIN
    ...
END

# 删除后
CREATE TRIGGER tri_after_delete_tb1 AFTER DELETE ON tb1 FOR EACH ROW
BEGIN
    ...
END

# 更新前
CREATE TRIGGER tri_before_update_tb1 BEFORE UPDATE ON tb1 FOR EACH ROW
BEGIN
    ...
END

# 更新后
CREATE TRIGGER tri_after_update_tb1 AFTER UPDATE ON tb1 FOR EACH ROW
BEGIN
    ...
END
```

```
delimiter //
CREATE TRIGGER tri_before_insert_tb1 BEFORE INSERT ON tb1 FOR EACH ROW
BEGIN

IF NEW. NAME == 'alex' THEN
    INSERT INTO tb2 (NAME)
VALUES
    ('aa')
END
END//
delimiter ;
```

```
delimiter //
```

--bajie\_new

### 3. Re:MySQL练习题参考答案

14、查询和“002”号的同学学习的课程完全相同的其他同学学号和姓名； select sid,sname from (select sid,sname from student where sid i...

--BobAylIn

### 4. Re:MySQL练习题参考答案

13、查询至少学过学号为“001”同学所有课的其他同学学号和姓名； 需要考虑这种情况，001同学选了1、2、4，002同学选了1、2、3。这种情况，两位同学选课有交集，count(选课数)也一样，然是...

--BobAylIn

5. Re:人生没有白走的路，每一步都算数 后续来了吗

--小鱼仔。

```
CREATE TRIGGER tri_after_insert_tb1 AFTER INSERT ON tb1 FOR EACH ROW
BEGIN
    IF NEW. num = 666 THEN
        INSERT INTO tb2 (NAME)
        VALUES
            ('666'),
            ('666') ;
    ELSEIF NEW. num = 555 THEN
        INSERT INTO tb2 (NAME)
        VALUES
            ('555'),
            ('555') ;
    END IF;
END//
delimiter ;
```

特别的：NEW表示即将插入的数据行，OLD表示即将删除的数据行。

## 2、删除触发器

```

DROP TRIGGER tri_after_insert_tb1;
```

## 3、使用触发器

触发器无法由用户直接调用，而知由于对表的【增/删/改】操作被动引发的。

```

insert into tb1(num) values(666)
```

## 存储过程

存储过程是一个SQL语句集合，当主动去调用存储过程时，其中内部的SQL语句会按照逻辑执行。

### 1、创建存储过程

```

-- 创建存储过程

delimiter //
create procedure p1()
BEGIN
    select * from t1;
END//
delimiter ;

-- 执行存储过程

call p1()
```

对于存储过程，可以接收参数，其参数有三类：

- in 仅用于传入参数用
- out 仅用于返回值用
- inout 既可以传入又可以当作返回值

```

-- 创建存储过程
delimiter \
```

```

create procedure p1(
    in i1 int,
    in i2 int,
    inout i3 int,
    out r1 int
)
BEGIN
    DECLARE temp1 int;
    DECLARE temp2 int default 0;

    set temp1 = 1;

    set r1 = i1 + i2 + temp1 + temp2;

    set i3 = i3 + 100;

end\\
delimiter ;

-- 执行存储过程
set @t1 =4;
set @t2 = 0;
CALL p1 (1, 2 ,@t1, @t2);
SELECT @t1,@t2;

```



```

delimiter //
create procedure p1()
begin
    select * from v1;
end //
delimiter ;

```



```

delimiter //
create procedure p2(
    in n1 int,
    inout n3 int,
    out n2 int,
)
begin
    declare temp1 int ;
    declare temp2 int default 0;

    select * from v1;
    set n2 = n1 + 100;
    set n3 = n3 + n1 + 100;
end //
delimiter ;

```



```

delimiter \\
create PROCEDURE p1(
    OUT p_return_code tinyint
)
BEGIN
    DECLARE exit handler for sqlexception

```

```

BEGIN
    -- ERROR
    set p_return_code = 1;
    rollback;
END;

DECLARE exit handler for sqlwarning
BEGIN
    -- WARNING
    set p_return_code = 2;
    rollback;
END;

START TRANSACTION;
DELETE from tb1;
insert into tb2(name) values('seven');

COMMIT;

-- SUCCESS
set p_return_code = 0;

END\\
delimiter ;

```

```

delimiter //
create procedure p3()
begin
    declare ssid int; -- 自定义变量1
    declare sname varchar(50); -- 自定义变量2
    DECLARE done INT DEFAULT FALSE;

    DECLARE my_cursor CURSOR FOR select sid,sname from s
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TR

    open my_cursor;
    xxoo: LOOP
        fetch my_cursor into ssid,sname;
        if done then
            leave xxoo;
        END IF;
        insert into teacher(tname) values(sname);
    end loop xxoo;
    close my_cursor;
end //
delimter ;

```

```

delimiter \\
CREATE PROCEDURE p4 (
    in nid int
)
BEGIN
    PREPARE prod FROM 'select * from student where sid >
EXECUTE prod USING @nid;
DEALLOCATE prepare prod;

```

```
END\\
delimiter ;
```

## 2、删除存储过程

```
drop procedure proc_name;
```

## 3、执行存储过程

```
-- 无参数
call proc_name()

-- 有参数, 全in
call proc_name(1,2)

-- 有参数, 有in, out, inout
set @t1=0;
set @t2=3;
call proc_name(1,2,@t1,@t2)
```

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
import pymysql

conn = pymysql.connect(host='127.0.0.1', port=3306, user='root', passwd='123')
cursor = conn.cursor(cursor=pymysql.cursors.DictCursor)

# 执行存储过程
cursor.callproc('p1', args=(1, 22, 3, 4))

# 获取执行完存储的参数
cursor.execute("select @_p1_0,@_p1_1,@_p1_2,@_p1_3")
result = cursor.fetchall()

conn.commit()
cursor.close()
conn.close()

print(result)
```

## 函数

MySQL中提供了许多内置函数，例如：

```
CHAR_LENGTH(str)

返回值为字符串str 的长度，长度的单位为字符。一个多字节字符算作一个单字符。
对于一个包含五个二字节字符集， LENGTH() 返回值为 10， 而CHAR_LENGTH() 的返回值为 5。

CONCAT(str1,str2,...)

字符串拼接
如有任何一个参数为NULL， 则返回值为 NULL。

CONCAT_WS(separator,str1,str2,...)

字符串拼接（自定义连接符）
```

CONCAT\_WS() 不会忽略任何空字符串。（然而会忽略所有的 NULL）。

CONV(N,from\_base,to\_base)

进制转换

例如：

SELECT CONV('a',16,2); 表示将 a 由16进制转换为2进制字符串表示

FORMAT(X,D)

将数字X 的格式写为 '#,###,###.##',以四舍五入的方式保留小数点后 D 位, 并将结果

例如：

SELECT FORMAT(12332.1,4); 结果为: '12,332.1000'

INSERT(str,pos,len,newstr)

在str的指定位置插入字符串

pos: 要替换位置其实位置

len: 替换的长度

newstr: 新字符串

特别的：

如果pos超过原字符串长度, 则返回原字符串

如果len超过原字符串长度, 则由新字符串完全替换

INSTR(str,substr)

返回字符串 str 中子字符串的第一个出现位置。

LEFT(str,len)

返回字符串str 从开始的len位置的子序列字符。

LOWER(str)

变小写

UPPER(str)

变大写

LTRIM(str)

返回字符串 str , 其引导空格字符被删除。

RTRIM(str)

返回字符串 str , 结尾空格字符被删去。

SUBSTRING(str,pos,len)

获取字符串子序列

LOCATE(substr,str,pos)

获取子序列索引位置

REPEAT(str,count)

返回一个由重复的字符串str 组成的字符串, 字符串str的数目等于count 。

若 count <= 0,则返回一个空字符串。

若str 或 count 为 NULL, 则返回 NULL 。

REPLACE(str,from\_str,to\_str)

返回字符串str 以及所有被字符串to\_str替代的字符串from\_str 。

REVERSE(str)

返回字符串 str , 顺序和字符顺序相反。

RIGHT(str,len)

从字符串str 开始, 返回从后边开始len个字符组成的子序列

SPACE(N)

返回一个由N空格组成的字符串。

SUBSTRING(str,pos) , SUBSTRING(str FROM pos) SUBSTRING(str,pos,len) , SUBSTRING(str,pos,LEN) 不带有len 参数的格式从字符串str返回一个子字符串, 起始于位置 pos。带有len参数的格式从字符串str返回一个子字符串, 起始于位置 pos, 长度为len。

```
mysql> SELECT SUBSTRING('Quadratically',5);
-> 'ratically'
```

```
mysql> SELECT SUBSTRING('foobarbar' FROM 4);
-> 'barbar'
```

```
mysql> SELECT SUBSTRING('Quadratically',5,6);
-> 'ratica'
```

```
mysql> SELECT SUBSTRING('Sakila', -3);
-> 'ila'

mysql> SELECT SUBSTRING('Sakila', -5, 3);
-> 'aki'

mysql> SELECT SUBSTRING('Sakila' FROM -4 FOR 2);
-> 'ki'
```

TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str) TRIM(remstr FROM]  
返回字符串 str , 其中所有remstr 前缀和/或后缀都被删除。若分类符BOTH、LEADI

```
mysql> SELECT TRIM(' bar ');
-> 'bar'

mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
-> 'barxxx'

mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
-> 'bar'

mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxyz');
-> 'barx'
```

更多函数: [中文猛击这里](#) OR [官方猛击这里](#)

### 1、自定义函数

```
delimiter \\  
create function f1(  
    i1 int,  
    i2 int)  
returns int  
BEGIN  
    declare num int;  
    set num = i1 + i2;  
    return(num);  
END \\  
delimiter ;
```

### 2、删除函数

```
drop function func_name;
```

### 3、执行函数

[View Code](#)

## 事务

事务用于将某些操作的多个SQL作为原子性操作，一旦有某一个出现错误，即可回滚到原来的状态，从而保证数据库数据完整性。

```
delimiter \\  
create PROCEDURE p1(  
    OUT p_return_code tinyint
```



```

)
BEGIN
  DECLARE exit handler for sqlexception
BEGIN
  -- ERROR
  set p_return_code = 1;
  rollback;
END;

DECLARE exit handler for sqlwarning
BEGIN
  -- WARNING
  set p_return_code = 2;
  rollback;
END;

START TRANSACTION;
  DELETE from tb1;
  insert into tb2(name) values ('seven');
COMMIT;

-- SUCCESS
set p_return_code = 0;

END\\
delimiter ;

```



```

1 | set @i =0;
2 | call p1(@i);
3 | select @i;

```

## 索引

索引，是数据库中专门用于帮助用户快速查询数据的一种数据结构。类似于字典中的目录，查找字典内容时可以根据目录查找找到数据的存放位置，然后直接获取即可。

1						30			
2									
3			10					40	
4									
5		5		15		35			66
6									
7	1	6	11	19	21	39	55	100	

MySQL中常见索引有：

- 普通索引
- 唯一索引
- 主键索引
- 组合索引

### 1、普通索引

普通索引仅有一个功能：加速查询



```

create table in1(
  nid int not null auto_increment primary key,
  name varchar(32) not null,
  email varchar(64) not null,
  extra text,
  index ix_name (name)
)

```



```
create index index_name on table_name(column_name)
```



```
drop index index_name on table_name;
```



```
show index from table_name;
```

注意：对于创建索引时如果是BLOB 和 TEXT 类型，必须指定length。



```
create index ix_extra on inl(extra(32));
```

## 2、唯一索引

唯一索引有两个功能：加速查询 和 唯一约束（可含null）



```
create table inl(  
    nid int not null auto_increment primary key,  
    name varchar(32) not null,  
    email varchar(64) not null,  
    extra text,  
    unique ix_name (name)  
)
```



```
create unique index 索引名 on 表名(列名)
```



```
drop unique index 索引名 on 表名
```

## 3、主键索引

主键有两个功能：加速查询 和 唯一约束（不可含null）



```
create table inl(  
    nid int not null auto_increment primary key,  
    name varchar(32) not null,  
    email varchar(64) not null,  
    extra text,  
    index ix_name (name)  
)
```

OR

```
create table inl(  
    nid int not null auto_increment,  
    name varchar(32) not null,  
    email varchar(64) not null,  
    extra text,  
    primary key(nid),  
    index ix_name (name)  
)
```



```
alter table 表名 add primary key(列名);
```



```
alter table 表名 drop primary key;
alter table 表名 modify 列名 int, drop primary key;
```

#### 4、组合索引

组合索引是将n个列组合成一个索引

其应用场景为：频繁的同时使用n列来进行查询，如：where n1 = 'alex' and n2 = 666。

```
create table in3(
    nid int not null auto_increment primary key,
    name varchar(32) not null,
    email varchar(64) not null,
    extra text
)
```

```
create index ix_name_email on in3(name,email);
```

如上创建组合索引之后，查询：

- name and email -- 使用索引
- name -- 使用索引
- email -- 不使用索引

注意：对于同时搜索n个条件时，组合索引的性能好于多个单一索引合并。

### 其他

#### 1、条件语句

```
delimiter \
CREATE PROCEDURE proc_if ()
BEGIN

    declare i int default 0;
    if i = 1 THEN
        SELECT 1;
    ELSEIF i = 2 THEN
        SELECT 2;
    ELSE
        SELECT 7;
    END IF;

END\
delimiter ;
```

#### 2、循环语句

```
delimiter \
CREATE PROCEDURE proc_while ()
BEGIN

    DECLARE num INT ;
    SET num = 0 ;
    WHILE num < 10 DO
```

```
SELECT
    num ;

SET num = num + 1 ;

END WHILE ;
```

```
END\\
delimiter ;
```



```
delimiter \\
CREATE PROCEDURE proc_repeat ()
BEGIN

    DECLARE i INT ;
    SET i = 0 ;
    repeat
        select i;
        set i = i + 1;
        until i >= 5
    end repeat;

END\\
delimiter ;
```



```
BEGIN

declare i int default 0;
loop_label: loop

    set i=i+1;
    if i<8 then
        iterate loop_label;
    end if;
    if i>=10 then
        leave loop_label;
    end if;
    select i;
end loop loop_label;
```

```
END
```



### 3、动态执行SQL语句



```
delimiter \\
DROP PROCEDURE IF EXISTS proc_sql \\
CREATE PROCEDURE proc_sql ()
BEGIN
    declare p1 int;
    set p1 = 11;
    set @p1 = p1;

    PREPARE prod FROM 'select * from tb2 where nid > ?';
    EXECUTE prod USING @p1;
    DEALLOCATE prepare prod;

END\\
```

```
delimiter ;
```



作者：武沛齐  
出处：<http://www.cnblogs.com/wupeiqi/>  
本文版权归作者和博客园共有，欢迎转载，但未经作者同意必须保留此段声明，且在文章页面明显位置给出原文连接。

好文要顶

关注我

收藏该文



武沛齐  
关注 - 44  
粉丝 - 9974

+加关注

18

0

posted @ 2016-07-28 07:12 武沛齐 阅读(22936) 评论(3) 编辑 收藏

评论列表

#1楼 2018-10-17 13:52 Uncle\_Drew [回复](#) [引用](#)  
索引补充: <https://www.cnblogs.com/wupeiqi/articles/5716963.html>  
[支持\(0\)](#) [反对\(0\)](#)

#2楼 2019-01-02 01:42 扫驴 [回复](#) [引用](#)  
武大师，最后一个，通过存储过程防sql注入，到底是怎么实现的啊，我来来回回看了好几遍你4期的讲解视频，都没搞懂这一块是什么意思  
[支持\(0\)](#) [反对\(0\)](#)

#3楼 2019-12-03 15:01 小粉优化大师 [回复](#) [引用](#)  
触发器的示例不能用，应该修正为  
delimiter //  
CREATE TRIGGER tri\_before\_insert\_tb1 BEFORE INSERT ON tb1 FOR EACH ROW  
BEGIN  
  
IF (NEW.name= 'alex') THEN  
INSERT INTO tb2 (name)  
VALUES  
(  
'aa');  
END IF;  
END//  
delimiter ;  
  
[支持\(0\)](#) [反对\(0\)](#)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

发表评论

[编辑](#) [预览](#)

B

支持 Markdown

[提交评论](#) [退出](#) [订阅评论](#)

[Ctrl+Enter快捷键提交]

【推荐】超50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库

【推荐】了不起的开发者，挡不住的华为，园子里的品牌专区

【推荐】30+视频&10+案例纵横文件与IO领域 | Java开发者高级应用站

#### 相关博文：

- Python开发【第十八篇】：MySQL（二）
  - 【Python之路】第十八篇--MySQL（一）
  - Python开发【第十八篇】：MySQL（二）
  - python学习[第十八篇]函数二（未完）
  - Python开发【第十八篇】：MySQL（二）
- » 更多推荐...

#### 最新 IT 新闻：

- 马斯克的Neuralink 是先知的指引还是无知的妄想？
  - 2020年度国家“杰出青年”公布，21人计算机领域贡献突出
  - 腾讯不想把半条命给合作伙伴了？
  - 听小米员工讲述他们所亲历的小米10年
  - 一边退场一边上市 新造车进入“季后赛”
- » 更多新闻...