在Mac或Linux上，需要编辑MySQL的配置文件，把数据库默认的编码全部改为UTF-8？所以最好使用前先看下编码标准吧

增加字段：

alter table 表名 ADD 字段 类型 约束 [默认值 注释]

ALTER TABLE video ADD category\_id int(11) unsigned not null DEFAULT '0' COMMENT '视频分类id';

在column1后增加字段

ALTER TABLE 表名 ADD 字段名 字段类型 AFTER column1;

在表的开始位置增加字段

ALTER TABLE user ADD id INT(10) FIRST;

删除字段

alter table t1 drop column age;

修改字段名：

alter table 表名 rename column A to B

ALTER TABLE video RENAME COLUMN category\_id TO cid;

修改字段类型：

alter table 表名 modify column 字段名 类型 约束 [默认值, 注释];

ALTER TABLE video MODIFY COLUMN category\_id smallint(5) unsigned not null DEFAULT '0' COMMENT '视频分类id';

修改字段默认值

alter table 表名 alter column 字段名 drop default; --(若本身存在默认值，则先删除)

alter table 表名 alter column 字段名 set default 默认值; --(若本身不存在则可以直接设定)

ALTER TABLE video ALTER COLUMN sort SET DEFAULT '50';

修改表名

alter table user rename new\_user

删除表内数据

delete from student where T\_name = "张三";

清除表内数据，保存表结构

truncate table student;

删除表

drop table student;

* 查看数据库**编码格式**

mysql> show variables like 'character\_set\_database';

* 查看数据表的编码格式

mysql> show create table <表名>;

* 创建数据库时指定数据库的字符集

mysql>create database <数据库名> character set utf8;

* 修改数据库的编码格式

mysql>alter database <数据库名> character set utf8;

* 修改数据表格编码格式

mysql>alter table <表名> character set utf8;

* 修改字段编码格式

mysql>alter table <表名> change <字段名> <字段名> <类型> character set utf8;

mysql>alter table user change username username varchar(20) character set utf8 not null;

我建表的时候没有设置默认编码为utf8，把数据库、表格、字段的编码格式都改了以后才生效

* MySQL 获得当前时间戳函数：current\_timestamp, current\_timestamp()

SELECT NOW(),CURDATE(),CURTIME()

* CREATE **TRIGGER** trigger\_name trigger\_time trigger\_event ON tb\_name FOR EACH ROW trigger\_stmt

trigger\_name：触发器的名称

tirgger\_time：触发时机，为BEFORE或者AFTER

trigger\_event：触发事件，为INSERT、DELETE或者UPDATE

tb\_name：表名

trigger\_stmt：触发器的程序体，可以是一条SQL语句或者是用BEGIN和END包含的多条语句

所以可以说MySQL创建以下六种触发器：

BEFORE INSERT BEFORE DELETE BEFORE UPDATE

AFTER INSERT AFTER DELETE AFTER UPDATE

DELIMITER //

CREATE TRIGGER t1 BEFORE INSERT ON client FOR EACH ROW

BEGIN

END //

DELIMITER ;

* 查看触发器

SHOW TRIGGERS;

* 删除触发器

DROP TRIGGER t1;

存储程序：

1.存储过程procedure

2.存储函数function

创建存储过程

CREATE PROCEDURE sp\_name ( [proc\_parameter] ) routine\_body

proc\_parameter参数列表

形式为[ IN | OUT| INOUT] param\_name type

mysql> DELIMITER //

mysql> Create procedure p1(OUT param1 INT)

-> begin

-> select count(\*) INTO param1 from clients;

-> end //

Query OK, 0 rows affected (0.39 sec)

mysql> dilemiter;

delimiter 作用将mysql结束符设置为//注意是斜杠不是反斜杠。因为mysql默认语句结束符号为 ; ，为了避免与存储过程中SQL语句冲突，需要使用Delimiter改变储存过程的结束符，以end //结束存储过程。delimiter; 恢复默认结束符

创建存储函数

DELIMITER //

CREATE FUNCTION p2(id INT)

RETURNS CHAR(10)

BEGIN

RETURN (SELECT name FROM CLIENTS WHERE cid=id);

END //

delimiter ;

调用存储过程

CALL p1( 100, num);

num如果是一个OUT的参数，结果就存在num里了

调用存储函数

SELECT p2(101);

删除

DROP FUNCTION p2;

DROP PROCEDURE p1;

可以在存储过程中使用的局部变量

DECLARE var1 int DEFAULT 100;

SET var1=200;//使用Set赋值

SELECT cid INTO var1 FROM clients where name='zhangxianseng';

光标Cursor，网上叫游标

DECLARE c1 CURSOR FOR SELECT id, name FROM clients;

打开光标

OPEN c1;

使用

FETCH c1 INTO var1;

关闭光标

CLOSE c1

貌似只能在函数中使用

流程控制语句

* IF var1 IS NULL

THEN SELECT 'var1 is null';

ELSE SELECT 'var1 is not null';

END IF;

* CASE var1

WHEN 100 THEN SELECT 'var1 is 100';

WHEN 200 THEN SELECT 'var1 is 200';

ELSE SELECT 'var1 is not 100 or 200';

END CASE

DECLARE id INT DEFAULT 0;

* l1 : LOOP

SET id=id+1;

IF id>=10 THEN LEAVE l1;

END LOOP l1;

leave 可以退出任何被标注的流程控制构造

ITERATE label;

出现在LOOP、REPEAT、WHILE语句内。相当于continue应该

Mysql变量

1.SET @sum=0;

会话变量。在当前客户端中有效

变量名必须是@开头

2.DECLARE var1 INT DEFAULT 199;

局部变量，只能用在begin和end之间，end之后没了

变量名不能有@

1. 数据定义语言(DDL ，Data Defintion Language)语句：数据定义语句，用于定义不同的数据段、数据库、表、列、索引等。常用的语句关键字包括create、drop、alter等。
2. 数据操作语言(DML ， Data Manipulation Language)语句：数据操纵语句，用于添加、删除、更新和查询数据库记录，并检查数据的完整性。常用的语句关键字主要包括insert、delete、update和select等。
3. 数据控制语言(DCL， Data Control Language)语句：数据控制语句，用于控制不同数据段直接的许可和访问级别的语句。这些语句定义了数据库、表、字段、用户的访问权限和安全级别。主要的语句关键字包括grant、revoke等。

**索引**

索引用于快速找出在某个列中有一特定值的行。

索引越多写入，修改的速度越慢。因为写入修改数据时也要修改索引

* 普通索引

创建索引

CREATE INDEX indexName ON mytable(username);

修改表结构(添加索引)

ALTER table tableName ADD INDEX indexName(columnName)

删除索引的语法

DROP INDEX [indexName] ON mytable;

* 唯一索引

索引列的值必须唯一，但允许有空值。如果是组合索引，则列值的组合必须唯一

创建索引

CREATE UNIQUE INDEX indexName ON mytable(username(length))

修改表结构

ALTER table mytable ADD UNIQUE [indexName] (username)

* 全文索引

全文索引是目前搜索引擎使用的一种关键技术。它能够利用"分词技术"等多种算法智能分析出文本文字中关键字词的频率及重要性，然后按照一定的算法规则智能地筛选出我们想要的搜索结果

创建article表，其中有主键ID(id)、文章标题(title)、文章内容(content)三个字段

在title和content两个列上创建全文索引

CREATE TABLE article (

id INT UNSIGNED AUTO\_INCREMENT NOT NULL PRIMARY KEY,

title VARCHAR(200),

content TEXT,

FULLTEXT (title, content)

);

在article表的title和content列中全文检索指定的查询字符串的SQL语句：

SELECT \* FROM article WHERE MATCH(title, content) AGAINST('查询字符串')

* MySQL自带的全文索引只能用于数据库引擎为MyISAM的数据表。此外，MySQL自带的全文索引只能对英文进行全文检索，目前无法对中文进行全文检索。
* 目前，使用MySQL自带的全文索引时，如果查询字符串的长度过短将无法得到期望的搜索结果。MySQL全文索引所能找到的词的默认最小长度为4个字符。另外，如果查询的字符串包含停止词，那么该停止词将会被忽略。
* 尽量先创建表并插入所有数据后再创建全文索引，而不要在创建表时就直接创建全文索引，因为前者比后者的全文索引效率要高。
* 主键索引

添加主键的同时自动添加索引

ALTER TABLE money ADD PRIMARY KEY(id);

创建表时声明索引：  
INDEX [索引名] (字段)  
FULLTEXT [索引名] (字段)  
UNIQUE[索引名] (字段)

中括号中的索引名，代表可选

CREATE TABLE test (  
    id INT NOT NULL ,   
     username VARCHAR(20) NOT NULL ,   
     password INT NOT NULL ,   
     content INT NOT NULL ,   
     PRIMARY KEY (id),   
     INDEX pw (password),   
     UNIQUE (username),  
    FULLTEXT (content)  
 )

显示索引信息

SHOW INDEX 命令列出表中的相关的索引信息。添加 \G 格式化输出信息。

SHOW INDEX FROM table\_name; \G

查询单个字段不重复记录

select distinct age from student;

* 结果集排序

select 字段 from 表 order by 字段 排序关键词

select name, age, id from student order by id desc;

asc 升序 ascend

desc 降序排列 descend

select id, name, age from student order by id desc, age asc;

id一样就根据age排

* 结果集限制

select \* from students limit 5;

显示5个学生

select id,name from student order by balance desc limit 5;

* 结果集区间选择

select 字段 from 表 limit 偏移量,数量

select \* from student limit 0,3;

从第一条开始取三条记录

* 第一条记录为0。

select \* from student limit 3,3;

* 统计类函数
* sum
* count
* max
* min
* avg

mysql> select count(id) as zongshu from money;

+---------+

| zongshu |

+---------+

| 12 |

+---------+

1 row in set (0.00 sec)

* 分组

select \* from student group by age;

按照年龄分组

年龄字段相同的不显示

mysql> select count(1), age from student group by age;

统计不同年龄学生的人数

count(1)和count(\*)和count(999)都显示一样

如果student有字段score，可以为null，那么

select count(\*) from student; /\* 5 \*/

select count(score) from student; /\* 3 \*/

这样写不统计score为null的记录

select count(\*), name from student group by age with rollup;

+----------+----------+

| count(\*) | name |

+----------+----------+

| 9 | 林然 |

| 3 | Jennifer |

| 12 | Jennifer | /\*加个with rollup多了一行这个, 统计总数\*/

+----------+----------+

* 结果过滤

having 是筛选组,而where是筛选记录

select count(\*) as result, age from student group by age having result > 2;

顺序

SELECT

[字段1 [as 别名1],[函数(字段2) ,]......字段n]

FROM 表名

[WHERE where条件]

[GROUP BY 字段]

[HAVING where\_contition]

[order 条件]

[limit 条件]

# Mysql

## 安装

用ZIP解压缩安装时，需要初始化数据目录。

太复杂了懒得看了我吐了

* SQL语言关键字不区分大小写！但是，不同的数据库，对于表名和列名，有的数据库区分大小写，有的数据库不区分大小写。同一个数据库，有的在Linux上区分大小写，有的在Windows上不区分大小写。习惯上SQL关键字都大写，表名列名都小写。
* MySQL本身实际上只是一个SQL接口，它的内部还包含了多种数据引擎，常用的包括：InnoDB、MyISAM
* MySQL接口和数据库引擎的关系就好比浏览器和浏览器引擎（IE引擎或Webkit引擎）的关系。使用MySQL时，不同的表还可以使用不同的数据库引擎。InnoDB最常用

基于MySQL的开源版本，又衍生出了各种版本：

* MariaDB 由MySQL的创始人创建的一个开源分支版本，使用XtraDB引擎。
* Aurora 由Amazon改进的一个MySQL版本
* PolarDB 由Alibaba改进的一个MySQL版本，专门提供给在阿里云托管的MySQL用户
* MySQL Client是一个命令行客户端，可以通过MySQL Client登录MySQL Server，真正的MySQL服务器程序是mysqld，在后台运行。。现在基本上是通过Workbench登录MySQL Server

配制好环境变量后，或者进入安装目录C:\Program Files\MySQL\MySQL Server 8.0\bin，输入

mysql -u root -p

再输入密码就连上了MySQL Server，同时提示符变为mysql>：

输入exit断开与MySQL Server的连接

在MySQL Client中输入的SQL语句通过TCP连接发送到MySQL Server。默认端口号是3306，即如果发送到本机MySQL Server，地址就是127.0.0.1:3306。

可以通过MySql Client远程链接MySQL Server，如果远程MySql Server的IP地址是10.0.1.99，那么就使用-h指定IP或域名：

mysql -h 10.0.1.99 -u root -p

## 关系模型

关系模型本质上就是若干个存储数据的二维表

表的每一行称为记录Record

表的每一列称为字段Column

一条记录由多个字段组成

## 数据库

创建数据库

CREATE DATABASE test1

CREATE DATABASE IF NOT EXISTS test1;

如果不存在，则创建

切换

USE test;

列出所有数据库

SHOW DATABASES;

## 数据表

* 建表

CREATE TABLE classes (

id BIGINT NOT NULL AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

PRIMARY KEY (id)

)DEFAULT CHARSET=utf8;

主键还可以这么声明

id BIGINT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

建表时选择引擎

CREATE TABLE classes (

id BIGINT NOT NULL AUTO\_INCREMENT,

PRIMARY KEY (id)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

* 插入记录

INSERT INTO students (id, class\_id, name, gender, score) VALUES (1, 1, '小明', 'M', 90);

主键即使Auto\_Increment，也可以直接规定，插入。满足约束就行

一次性增加多条记录

INSERT INTO students (class\_id, name, gender, score) VALUES

(1, '大宝', 'M', 87),

(2, '二宝', 'M', 81);

* 删表

DROP TABLE IF EXISTS classes;

* 数据类型

INT 整型 4字节整数类型，范围约+/-21亿

BIGINT 长整型 8字节整数类型，范围约+/-922亿亿

REAL 浮点型 4字节浮点数，范围约+/-1038

DOUBLE 浮点型 8字节浮点数，范围约+/-10308

DECIMAL(M,N) 高精度小数 指定精度的小数，例如，DECIMAL(20,10)表示一共20位，其中小数10位，通常用于财务计算

CHAR(N) 定长字符串

VARCHAR(N) 变长字符串

BOOLEAN 布尔类型 存储True或者False

DATE 日期类型 2018-06-22

TIME 时间类型 12:20:59

DATETIME 日期+时间 2018-06-22 12:20:59

* REAL又可以写成FLOAT(24)
* BIGINT UNSIGNED NOT NULL AUTO\_INCREMENT UNIQUE 又可以写成SERIAL
* 列出当前数据库中所有表

SHOW TABLES;

* 查看一个表的结构

DESC students;

查看创建表的SQL语句

SHOW CREATE TABLE students;

## 主键

对某一列添加一个唯一约束，不是主键也保证唯一

ALTER TABLE students

ADD CONSTRAINT uni\_name UNIQUE (name);

## 外键

* 添加外键

ALTER TABLE table1

ADD CONSTRAINT constraint1

FOREIGN KEY (col1)

REFERENCES table2 (col2);

* 约束名称constraint1可以是任意名称
* FOREIGN KEY指定col1作为外键
* REFERENCES table2 (col2) 指定这个外键将关联到table2表的col2列，col2是table2的主键
* 通过定义外键约束，关系数据库可以保证无法插入无效的数据。如果table2表不存在col2=99的记录，table1表就无法插入col1=99的记录。
* 外键约束会降低数据库的性能，大部分互联网应用程序为了追求速度，并不设置外键约束，而是仅靠应用程序自身来保证逻辑的正确性。
* 删除外键

ALTER TABLE students DROP FOREIGN KEY fk\_class\_id;

删除外键约束并没有删除外键这一列。删除列是通过DROP COLUMN ...实现的。

## 索引

* 添加索引

ALTER TABLE students

ADD INDEX idx\_score (col1);

使用ADD INDEX index\_name (col1)就创建了一个名称为index\_name，使用列col1的索引。

索引如果有多列，可以在括号里依次写上

ALTER TABLE students

ADD INDEX idx\_score (col1, col2);

* 索引提高数据量很大的时候的查询速度。
* 索引是关系数据库中对某一列或多个列的值进行预排序的数据结构。通过使用索引，可以让数据库系统不必扫描整个表，而是直接定位到符合条件的记录，大大加快查询速度。
* 索引的效率取决于索引列的值是否散列，即该列的值如果越互不相同，那么索引效率越高。反过来，如果记录的列存在大量相同的值，例如gender列，大约一半的记录值是M，另一半是F，因此，对该列创建索引就没有意义。
* 可以对一张表创建多个索引。索引的优点是提高了查询效率，缺点是在插入、更新和删除记录时，需要同时修改索引，因此，索引越多，插入、更新和删除记录的速度就越慢。
* 对于主键，关系数据库会自动对其创建主键索引。使用主键索引的效率是最高的，因为主键会保证绝对唯一。
* 唯一索引

ALTER TABLE students

ADD UNIQUE INDEX uni\_name (name);

通过UNIQUE关键字我们就添加了一个唯一索引。

保证name列的值唯一，同时添加索引。身份证号、手机号可以用这个

## 查询

SELECT \* FROM table1

\*表示“所有列”

FROM表示将要从哪个表查询

SELECT \* FROM table1 WHERE col1 >= 80

条件查询

SELECT \* FROM table1 WHERE col1 >= 80 AND col2='你好'

AND表示与，同时满足

SELECT \* FROM table1 WHERE col1 >= 80 OR col2='你好'

OR表示或，满足一个条件即可

SELECT \* FROM table1 WHERE col1 <> 80

SELECT \* FROM table1 WHERE NOT col1 = 80

两种表示查询col1不等于80的方法

SELECT \* FROM students WHERE (score < 80 OR score > 90) AND gender = 'M';

加括号可以改变运算优先级

如果不加括号，条件运算按照NOT、AND、OR的优先级进行，即NOT优先级最高，其次是AND，最后是OR。

* 投影查询

SELECT col1, col2 FROM table1

只返回某些列的数据，而不是所有列的数据，让结果集仅包含指定列。

这种操作称为投影查询。

* 排序

查询结果通常根据主键排序

SELECT id, name, gender, score FROM students ORDER BY score;

这样可以通过其他字段排序

SELECT id, name, gender, score FROM students ORDER BY score DESC;

DESC表示降序。

默认的排序规则是ASC升序，即从小到大。ASC可以省略，即ORDER BY score ASC和ORDER BY score效果一样。

SELECT id, name, gender, score FROM students ORDER BY score DESC, gender;

如果score列有相同的数据，要进一步排序，可以继续添加列名。例如这句表示先按score列倒序，如果有相同分数的，再按gender列排序：

* ORDER BY子句要放到WHERE子句后面
* LIMIT OFFSET要放到ORDER BY 后边
* 分页

SELECT id, name FROM students LIMIT 3 OFFSET 0;

* 对结果集从0号记录开始，最多取3条。
* SQL记录集的索引从0开始。
* 如果要查询第2页，则设置OFFSET 3。好像只能显示一页，想看后一页看不了，有点像截取，只能查看第N-N+3的记录
* OFFSET超过了查询的最大数量并不会报错，而是得到一个空的结果集。
* OFFSET是可选的，如果只写LIMIT 15，那么相当于LIMIT 15 OFFSET 0。
* 在MySQL中，LIMIT 15 OFFSET 30还可以简写成LIMIT 30, 15。
* 聚合查询

SELECT COUNT(\*) FROM students;

查询一共有多少条记录

通常使用聚合查询时，应该给列名设置一个别名，便于处理结果。得出的结果有列名

SELECT COUNT(\*) num FROM students;

除了count，还有

SUM 合计值

AVG 平均值

MAX 最大值

MIN 最小值

SUM、AVG函数只能计算数值

SELECT AVG(score) FROM students;

SELECT COUNT(\*) num FROM students GROUP BY class\_id;

先按class\_id分组，然后计算

SELECT class\_id, AVG(score) num FROM students GROUP BY class\_id;

这样可以清楚的看到每个班级的平均分。有3个班，就有3个记录。但不能再SELECT后加name，因为任意一个分组中name是不同的，只有class\_id相同。所以只能查询分组的列

可以使用多个列进行分组

SELECT class\_id, gender, AVG(score) num FROM students GROUP BY class\_id, gender;

这样结果有6个记录，按班分组后再按照性别分组，分别算每个班男女平均分

* 多表查询

SELECT \* FROM table1, table2

返回table1和table2乘积。把table1的每一行，与table2的每一行拼在一起返回。结果行数为count(table1) \* count(table2)，列数为table1列数+table2列数

又叫笛卡尔查询。

如果table1和table2都有一个字段叫id，则可以通过设置别名区分

SELECT table1.id id1, table2.id id2 from table1, table2

还可以给表设置别名

SELECT t1.id id1, t2.id id2 from table1 t1, table2 t2

当然，设置别名在非多表查询的查询中也适用

* 连接查询

连接查询是另一种类型的多表查询。连接查询对多个表进行JOIN运算，就是先确定一个主表作为结果集，然后，把其他表的行有选择性地“连接”在主表结果集上。

SELECT table1.col1, table1.col2, table2.col3 FROM table1 INNER JOIN table2 on table1.col1=table2.col2

查询table1的col1和col2字段，只能显示这两个，如果想要同时显示table2的col3字段，而table2的col3字段可以通过table1.col1查询到，就可以这样写

* 先确定主表，仍然使用FROM table1的语法
* 再确定需要连接的表，使用INNER JOIN table2的语法
* 然后确定连接条件，使用ON <条件...>，这里的条件是table1.col1=table2.col2，表示table1表的col1列与table2表的col2列相同的行需要连接

SELECT ... FROM tableA JOIN tableB ON tableA.column1 = tableB.column2;

JOIN的种类有

* INNER JOIN 最常用
* LEFT OUTER JOIN 选出左表存在的记录。LEFT JOIN和LEFT OUTER JOIN效果一样
* RIGHT OUTER JOIN 选出右表存在的记录：
* FULL OUTER JOIN 选出左右表都存在的记录。貌似mysql不支持，可以通过union LEFT OUTER JOIN 和 RIGHT OUTER JOIN的结果得出

## 修改数据

UPDATE table1 SET col1='fffffff', col2=66 WHERE col3=1;

更新字段时可以使用表达式

UPDATE table1 SET col1=col1+1 WHERE col3=1;

删除

DELETE FROM students WHERE id=1;

## 修改表

新增一列

ALTER TABLE table1 ADD COLUMN col1 VARCHAR(10) NOT NULL;

要修改birth列，可以修改列名和数据类型

ALTER TABLE students CHANGE COLUMN birth birthday VARCHAR(20) NOT NULL;

要删除列，使用：

ALTER TABLE students DROP COLUMN birthday;

# MySQL with Python

MySQL Connector/Python

## Guidelines for Python Developers

* 不要在主脚本中硬编码hardcode与连接数据库有关的值。python有把这些值写在config.py的惯例
* 优化存储空间和磁盘IO optimize storage space and disk I/O的技术很重要。比如使用数字ID而不是字符串。
* 防止“SQL injection” attack

……………………

* 如果需要SSL支持,确保你的python安装是经过OpenSSL库编译过的

## Installation

pip install mysql-connector-python

## Connection

import mysql.connector

cnx = mysql.connector.connect(user='root', password='123321', host='127.0.0.1', database='world')

cnx.close()

处理错误：

import mysql.connector

from mysql.connector import errorcode

try:

cnx = mysql.connector.connect(user='root', database='world')

except mysql.connector.Error as err:

if err.errno == errorcode.ER\_ACCESS\_DENIED\_ERROR:

print("Something is wrong with your user name or password")

elif err.errno == errorcode.ER\_BAD\_DB\_ERROR:

print("Database does not exist")

else:

print(err)

exit()

else:

cnx.close()

else如果没有发生错误则执行

这种有点离谱哇，else里竟然用到了cnx。仔细想想也合理，因为没有错误表示try中的语句全部正常执行了

tmd到后来教程示例代码也不用try except了，肯定是自己都嫌麻烦

设置了database参数，就不用执行use语句了

cnx = mysql.connector.connect(user='joe', database='test')

字段 默认值

* host 127.0.0.1
* port 3306，mysql server的端口号
* use\_unicode True
* charset utf8
* connection\_timeout 无默认值
* autocommit False，需要手动调用commit()方法提交insert

可以用SSL登录

cnx.set\_config(\*\*dbconfig)

修改连接设置

## Creating Tables

All DDL (Data Definition Language) statements are executed using a handle structure known as a cursor.

创建和删除tables是一个expensive operation，需要避免一次又一次创建删除

temporary table是个例外，可以在一个应用中快速创建并删除

cursor = cnx.cursor()

创建数据库

cursor.execute("CREATE DATABASE {} DEFAULT CHARACTER SET 'utf8'".format('testDB'))

cursor.execute("USE {}".format('testDB'))

正确的写法是try use testDB，如果不存在，则创建。Example use database

创建表

TABLES = {}

TABLES['students'] = (

"CREATE TABLE students ("

"stu\_id int(11) NOT NULL AUTO\_INCREMENT, "

"birth\_date date NOT NULL, "

"name varchar(14) NOT NULL, "

"gender enum('M', 'F') NOT NULL,"

"PRIMARY KEY(stu\_id)"

")ENGINE=InnoDB"

)

cursor.execute(TABLES['students'])

好的写法是把所有的表声明语句都写好在TABLE对象中，然后用一个for循环执行

execute卸载try中，except捕获可能的错误，比如表已经存在

这种一行写不完，一个字符串写成多行的形式，在本教程中会很常见。效果完全等同于连在一起写

注意枚举类型

## Inserting Data

from datetime import date, datetime, timedelta

add\_stu = ("INSERT INTO students(birth\_date, name, gender) "

"VALUES (%s, %s, %s)")

stu1=(date(1998, 11, 9), 'Nfboys', "M")

cursor.execute(add\_stu, stu1)

cnx.commit()

如果使用一个transactional storage engine(事务存储引擎？)，比如InnoDB，必需在INSERT, DELETE 和 UPDATE 语句序列之后commit

add\_stu = ("INSERT INTO students (birth\_date, name, gender)"

"VALUES (%(ddd)s, %(nnn)s, %(ggg)s)")

stu2 = {

'ggg': "F",

'nnn': 'tingting',

'ddd': date(1650, 12, 12),

}

cursor.execute(add\_stu, stu2)

cnx.commit()

另一种方法

这种写法叫extended Python format codes

stu\_no = cursor.lastrowid

print(stu\_no)

在execute之后紧跟一句lastrowid，得到刚刚插入的记录的primary key

## Querying Data

query = ("SELECT stu\_id, birth\_date,gender FROM students "

"WHERE birth\_date BETWEEN %s AND %s")

date1 = date(1900, 1, 1)

date2 = date(2020, 12, 31)

cursor.execute(query, (date1, date2))

for (stu\_id, birth\_date, gender) in cursor:

print("stu\_id={}, birth\_date={}, gender={}".format(stu\_id, birth\_date, gender))

print(cursor.rowcount)

cursor.close()

cnx.close()

把cursor当做一个iterator，用for循环遍历得到的结果。

for循环可以引用多个变量

cursor用完也得close

Mysql Connector会自动把数据库DATETIME转为datetime.datetime类型

我明白了，cursor就是一个generator，在for的时候才执行，execute() select之后是不执行的

cursor.rowcount返回select的结果数量，或者INSERT/UPDATE语句影响的记录数量

* buffered cursors

默认情况，MySQL Connector不会buffer缓存查询结果。这样避免了过量的内存使用，如果数据量很大。如果你确定结果集合很小，完全可以一次处理，那么可以使用buffered cursors

curA = cnx.cursor(buffered=True)

或者在连接时指定buffer=true

cnx = mysql.connector.connect(user='joe', database='test', buffer=True)

还是用for循环读取结果

buffered cursors中，cursor.rowcount在execute select之后就是结果数量，比如6。default cursor中，cursor.rowcount在execute select之后是0，随着for循环进行而变成1,2,3...

## Connection Pooling

连接池打开若干个连接，在很多请求时处理线程安全。

连接池size大小在创建时指定，之后不能更改

可以有多个连接池，比如连接不同的MySQL server时可以用。

包含对多个MySQL服务器的连接的middleware

对MySQL服务器有多个永久的开放连接的网站

会从connection pooling受益

cnx = mysql.connector.connect(pool\_name = "nihao", pool\_size = 3)

在创建连接时指定连接池名和大小

pool\_size默认5，最大32

cnx.pool\_name获得连接池名称

后面的连接就可以只指定pool\_name，表示从这个连接池建立连接。其实已经不是建立新连接了，而是获取连接池中已经存在的连接

cnx2 = mysql.connector.connect(pool\_name="nihao")

如果这条语句重复4次，而连接池最大尺寸3，就会报Failed getting connection; pool exhausted错误

建立连接池后相当于同时建立了3个数据库连接，程序在进行过程中只使用这3个连接。程序退出后释放这些连接

cnx.close()

对于一个pooled connection，close方法实际上不关闭连接，而是让connection pool对于随后的连接请求有效

pooled connection不能使用connection.config()方法配置

## Appendix

### Example use database

Use database，create if not exist

def create\_database(cursor):

try:

cursor.execute(

"CREATE DATABASE {} DEFAULT CHARACTER SET 'utf8'".format(DB\_NAME))

except mysql.connector.Error as err:

print("Failed creating database: {}".format(err))

exit(1)

try:

cursor.execute("USE {}".format(DB\_NAME))

except mysql.connector.Error as err:

print("Database {} does not exists.".format(DB\_NAME))

if err.errno == errorcode.ER\_BAD\_DB\_ERROR:

create\_database(cursor)

print("Database {} created successfully.".format(DB\_NAME))

cnx.database = DB\_NAME

else:

print(err)

exit(1)