Note: MySQL

* 教程地址：

<https://www.runoob.com/mysql/mysql-tutorial.html>

* 学习目标：
* 背景知识
* 本地安装
* 创建数据库、数据表
* 数据操作：增删查改

# 基本知识

## 术语-数据库

* **关系数据库Relational Database**

关系数据库，是创建在关系模型基础上的数据库，是将数据间的关系以数据库表的形式加以表达，并将数据存储在表格中，以便于查询。

A relational database is a digital database based on the relational model of data.

* **关系数据库管理系统RDBMS**

关系数据库管理系统是管理关系数据库的数据库管理系统。

A software system used to maintain relational databases is a relational database management system (RDBMS).

* **结构化查询语言SQL(Structured Query Language)**

Structured Query Language is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS)

* **数据库Database**

**数据库**是按照数据结构来**组织、存储和管理**数据的仓库。

A **database** is an organized collection of data, generally stored and accessed electronically from a computer system.

* **数据表Datatable**

**表**是数据的矩阵。在一个数据库中的表看起来像一个简单的电子表格。

A **table** is a collection of related data held in a table format within a database. It consists of columns and rows.

A table can be considered a convenient representation of a relation

**表**是关系（元组或记录）的集合。

* **列Column/Field**

**列**(数据元素) 包含了相同类型的数据, 例如邮政编码的数据。

In a relational database, a **column** is a set of data values of a particular simple type, one value for each row of the database.

A column can also be called an attribute.

**列**是关系（元组或记录）的属性。

* **行Row**

**行**（**元组tuple**，或**记录**）是一组相关的数据，例如一条用户订阅的数据。

In the context of a relational database, a **row**—also called a **tuple**—represents a single, implicitly structured data item in a table.

Each row in a table represents a set of related data, and every row in the table has the same structure.

* **冗余data redundancy**

**存储两倍数据**，冗余降低了**性能**，但提高了数据的**安全性**。

Data redundancy is the existence of data that is additional to the actual data and permits correction of errors in stored or transmitted data.

在实际的数据库设计中，为提高数据库响应速度，减少响应时间，需要**冗余**，即增加数据库空间。简而言之，用空间换时间。

## 术语-数据库设计

### 第一范式First normal form(1NF)：字段不能再分

第一范式是指数据库表的**每一列都是不可分割的基本数据项**，同一列中**不能有多个值**，即实体中的某个属性不能有多个值或者**不能有重复的属性**。

注意，不是指数据表中某一属性不能有重复值，而是指每个列的值域都是由原子值组成，原子值不能再分。例如，在一个表里有一个“电话”字段，其值既有手机号，又有座机号 那么这样就不符合第一范式的要求了。此时应该将“电话”拆分为“手机”跟“座机” 这样一来就变成了一个符合第一范式的数据表了。

First normal form (1NF) is a property of a relation in a relational database.

### 第二范式(2NF): 组合主键时依赖性

第二范式是在第一范式的基础上建立起来的，即满足第二范式必须先满足第一范式。第二范式要求数据库表中**的每个实例或行必须可以被惟一的区分**。为实现区分，通常需要为表加上一个列，以存储各个实例的**惟一标识**，要求实体的属性完全依赖于主键。要讨论一个表符不符合第二范式的前提是，这个表的主键为组合主键，**如果不是组合主键那就谈不上符不符合第二范式了。**

总结：不能部分依赖，就是说当一个表有组合主键时，其他非主键的字段必须完全依赖于主键。

### 第三范式(3NF)：主键以外的字段必须依赖主键 而不能依赖其他字段

第三范式是要符合第二范式，第三范式就是一个数据表中不能存在其他数据表中的非主键字段，也就是说如果存在其他表中的字段，就一定得是那个表的主键。

假如一个表有有a b c三个字段，a—>b b—>c。 那么c与a就是传递依赖，这样就不符合第三范式。而且不管a能不能直接确定c都属于传递依赖，所以只有b跟c不依赖于a以外的任何字段才符合第三范式。

总结：不可以有传递依赖，就是说**主键以外的字段必须依赖主键 而不能依赖其他字段**。

## 术语-关系、元组、键

### 关系

若干集合笛卡尔积中有意义的元组的集合。笛卡尔积的有限子集。

关系的属性：关系名（表名）；属性列表（集合列表）和值域（集合可能取值范围）；属性间的数据依赖（完整性约束）。

* 例子：集合A是{a,b,c}，集合B是{x,y}。则集合A、B的笛卡尔积（所有关系）是{(a,x),(b,x),(c,x),(a,y),(b,y),(c,y)}。

而实际上A与B的关系是{(a,x),(c,x),(b,y)}，则该集合是所有关系的有限子集。

套用显示例子，A集合是学生，B集合是课程。则其笛卡尔积，就是可能的所有选课情况。

而学生实际的选课情况是其所有选课组合的子集。其选课情况就是学生集合和课程集合的关系。

### 多元组Tuple

* 数学上，n元组或多元组是对象个数有限的序列。元组由三部分组成：边界符、分隔符和元素。通常采用的边界符是小括号“()”，分隔符是逗号。

In mathematics, a tuple is a finite ordered list (sequence) of elements. An *n-tuple* is a sequence (or ordered list) of n elements, where n is a non-negative integer. There is only one 0-tuple, referred to as the **empty tuple**.

* 在数据库的关系模型，多元组通常被定义为从字段名到特定值的有限函数。其目的和在数学中一样，就是指出特定的实体，或那些包含特定部件且（或）具有特定性质的对象。

Relational databases may formally identify their rows (records) as tuples.



### 超键superkey

超键是在数据库关系模式设计中能够唯一标示多元组（“行”）的属性集合。

### 候选键candidate key

在关系模型中，候选键是某个关系变量的一组属性所组成的集合，它需要同时满足下列两个条件：

* + 这个属性集合始终能够**确保**在关系中能唯一标识元组
  + 在这个属性集合中找不出真子集能够满足条件(1)

满足第一个条件的属性集合称为***超键***，因此我们也可以把候选键定义为“***最小超键***”，也就是不含有多余属性的超键。

### 唯一键unique key

唯一键是候选键的子集。

从技术的角度来看，primary key和unique key有很多相似之处。但还是有以下区别：

* + 作为primary key的域／域组不能为Null。而unique key可以。
  + 一个表只能有一个primary key，但可有多个unique key。
  + 更大的区别在逻辑设计。primary key一般在逻辑设计中用作记录标识，这也是设置primary key的本意；而unique key只是为了保证域／域组的唯一性。

### 主键Primary key

主键是唯一的。一个数据表中只能包含一个主键。你可以使用主键来查询数据。

A primary key is a specific choice of a minimal set of attributes (columns) that uniquely specify a tuple (row) in a relation (table).

Informally, a primary key is "which attributes identify a record".

More formally, a primary key is a choice of candidate key (a minimal superkey); any other candidate key is an alternate key.

### 外键Foreign key

子数据表中出现的父数据表的主键，称为子数据表的外键。

In simpler words, a foreign key is a set of attributes that references a candidate key.

### 代理键surrogate key

当不适合用任何一个候选键作为主键时（如数据太长等），添加一个没有实际意义的键作为主键，这个键就是代理键。（如常用的序号1、2、3）

### 例子1：

元组(学生:"Harry", 课程:"数学课",分数:50)

* ***关系***：此时元组表示学生Harry选“数学课”，即表明学生集合、课程集合、分数集合的一种***关系***，对应现实中的***学生的课程成绩***，是学生集合、课程集合、分数集合的笛卡尔积的***子集***。
* ***关系属性***：关系名（表名）可以是“学生成绩”；属性列表是学生、课程、分数；域：字符串、整数。
* ***字段Field或属性Attribute*：**在该元组中，学生、课程、分数是元组的字段或属性。
* ***值Value***：“Harry”、“数学课”、50是对应字段或属性的值；
* ***域***：属性的取值范围的集合。集合中的所有元素都具有相同的数据类型。
* ***属性集或键Key***：元组中属性或字段的组合。这样的组合可以是多个，将所有的组合集合为一个集合，即属性集（键的集合）。即Set{(学生),(课程),(分数),(学生,课程),(学生,分数) ,(分数,课程) ,(学生,课程,分数)}。
* ***超键superkey***：在属性集中，即属性或字段的不同组合中，能够唯一标识元组（即一种关系）的属性集。超键是所有键的子集。

例如，当一个学生只能选一门课时，(学生)能唯一标识一个元组；学生和课程组合的键(学生,课程)，**也**能唯一标识一个元组；学生、课程、分数组合的键(学生,课程,分数)，**也**能唯一标识一个元组；此时超键是set{(学生), (学生,课程), (学生,课程,分数)}，是所有键（属性集）的子集

* ***候选键candidate key*：**不含有多余属性的超键，即含有最少属性且能唯一标识元组的属性集。候选键是超键的子集。只要满足要求，候选集可能有多个。

例如，set{(学生), (学生,课程), (学生,课程,分数)}是超键，但{(学生)}有最少属性且能唯一标识。所以{(学生)}是候选集。是超键的子集。

* ***主键Primary Key***：是人为地在可能多个的候选集中选定的一个，每个数据表只能选定一个。

例如，学生信息中，学号、身份证号都是候选键。在设计数据表时，人为地指定学号为数据表的主键。

* ***唯一键unique key***：是候选键的子集。如学生信息中，当学号和身份证号都是唯一键。一个数据表可以有一个以上的唯一键。
* ***代理键surrogate key***：当不方便指定元组中的任何候选键为主键时，人为地添加一个无意义的键，用于唯一地标识元组。常常是序号。
* ***外键foreign key***：例如在优秀学生表(子表）中，使用学生成绩（父表）的主键指代学生成绩表（父表）中的一个元组。

### 例子2

Customer（**Customer ID**, Tax ID, Name, Address, City, State, Zip, Phone）

Order（**Order No**, Customer ID, Invoice No, Date Placed, Date Promised, Terms, Status）

Order Line（**Order No**, **Order Line No**, Product Code, Qty）

Invoice（**Invoice No**, Customer ID, Order No, Date, Status）

Invoice Line（**Invoice No**, **Line No**, Product Code, Qty Shipped）

Product（**Product Code,** Product Description）

在这个设计中我们有六个***关系变量（数据表）***：Customer, Product, Order, Order Line, Invoice,和Invoice Line.

**粗体字有下划线**的属性是***候选键 (码)***。非粗体字有下划线的属性是***外键 (码)***。

通常任意选择一个***候选键 (码)***叫做***主键 (码)***并且优先于其他***候选键（码）***，它们也就被叫做***可选键 (码)***。

***候选键（码）***是强制元组不重复的唯一性标识符；否则关系就违背了集合的基本定义而成为是叫做包的东西了。***键 (码)***可以是复合的，就是说可以由多个属性组合而成。

## 术语-数据类型

## 术语-约束

# 安装

## 安装MySQL（略）

官网：

<https://dev.mysql.com/>

## 安装mysql-connector-python库（略）

pip命令：

pip install mysql-connector-python

文档：

<https://dev.mysql.com/doc/connector-python/en/>

导入库：

import mysql.connector

# 创建连接Connection

* 连接方法：
* 方法mysql.connector.connect()，；
* 创建mysql.connector.connection.MySQLConnection()类实例。

## 方法connect()

* connect()方法是在mysql模块下的connector模块中。所以需要引入模块mysql.connector。
* 对本地安装的mysql连接，最少需要给属性user、password赋值。

代码：

import mysql.connector  # 导入库

cnx = mysql.connector.connect(user='root',

                              password='Simonangel!23fong')

print(cnx)

cnx.close()

返回结果：



## 创建MySQLConnection()类实例

* 类MySQLConnection是在mysql模块下的connector模块的connection文件中，所以需要导入模块mysql.connector。
* 对本地安装的mysql连接，最少需要给属性user、password赋值。

代码：

from mysql.connector import (connection)  # 导入connection模块

# 创建MySQLConnection类的实例

cnx = connection.MySQLConnection(user='root',

                                 password='Simonangel!23fong')

print(cnx)

cnx.close()

返回结果：



## 使用字典设置连接参数

能够使用字典{}，运用方法connect()中的参数\*\* operator进行参数配置。

实例代码：

import mysql.connector  # 导入模块

config = {

    'user': 'root',

    'password': 'Simonangel!23fong'

}

cnx = mysql.connector.connect(\*\*config)

print(cnx)

cnx.close()

返回结果：



## 连接异常：errors.Error

* 连接器异常connection errors，由mysql模块下的connector模块的error.py文件处理，所以需要引入模块mysql.connector。
* 在error.py文件中Error类处理所有异常，其中的errno属性是异常代码，通常在抛出except中接住异常。
* mysql模块下的connector模块的errorcode.py文件，存储有所有异常代码。通常在处理异常时，通过异常代码判断异常的类型。

代码：

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

try:

    cnx = mysql.connector.connect(user='root',

                                  password='Simonangel!23fong',

                                  database='employ')

except mysql.connector.Error as err:  # 接住异常

    if err.errno == errorcode.ER\_ACCESS\_DENIED\_ERROR:  # 判断是否拒绝连接的异常

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

    elif err.errno == errorcode.ER\_BAD\_DB\_ERROR:    #判断是否数据库异常

        print("Database does not exist")

    else:

        print(err)

else:

    cnx.close()

返回结果：

当输入用户名或密码错误时，返回：



当用户名或密码正确时，返回：



## 总结

* 常用建立连接方式connect()+dict
* 导入模块mysql.connector

# 执行SQL语句execute

* 常见方法：cursor.execute()

## 执行查询语句：execute(sql)

* 常见步骤：

1.创建连接connection

2.创建游标connection.cursor

3.执行SQL语句cursor.execute(sql)

* 实例代码：

import mysql.connector  # 导入模块

sql = "SHOW DATABASES"  # 定义SQL语句

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

}

try:

    conn = mysql.connector.connect(\*\*config)  # 创建连接

    cursor = conn.cursor()  # 创建游标

    cursor.execute(sql)  # 执行SQL语句

    result = cursor.fetchall()  # 从游标中获取执行SQL语句返回的结果

    print(result)

except mysql.connector.Error as err:  # 接住异常

    print("Error: " + str(err))

else:

    cursor.close()  # 关闭游标

    conn.close()  # 关闭连接

返回结果：



## 执行查询语句：execute(sql,param)

* cursor.execute()的一种实现方式。
* 参数sql中使用%s或%(name)s作为占位符，
* 参数param，
  + 数据类型可以是元组()，当元组只含一个属性时，末尾必须以逗号结束。元组中对应的数据代入sql参数中的相应占位符。
  + 数据类型可以是字典，此时sql使用%(name)s，当中name对应字典中的name属性的值。

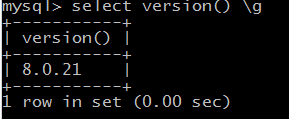
## 执行事务语句：cursor.execute(sql)+connection.commit()

# 服务器信息

## 获取当前MySQL版本

### CMD命令

*SELECT VERSION()*



### Python实例：

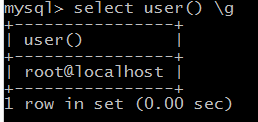
    cursor.execute('SELECT VERSION()')  # 执行SQL语句



## 获取当前用户名

### CMD命令

*select USER()*



### Python实例：

    cursor.execute('select user()')  # 执行SQL语句



# 数据库操作

## 创建数据库CREATE DATABASE

### SQL语句: 创建数据库

*CREATE DATABASE <db\_name>*

### Python实例 : 创建数据库

* 目标：创建一个名为my\_db的数据库
* 实例代码：

import mysql.connector  # 导入模块

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

}

try:

    conn = mysql.connector.connect(\*\*config)  # 创建连接

    cursor = conn.cursor()  # 创建游标

    cursor.execute("CREATE DATABASE my\_db")  # 执行SQL语句

except mysql.connector.Error as err:  # 接住异常

    print("Error: " + str(err))

else:

    cursor.close()  # 关闭游标

    conn.close()  # 关闭连接

* 目标验证：

使用SQL语句返回所有数据SHOW DATABASES。



### 细节注意

* SQL语句中的数据库名无需要用引号包围，否则会报错。

"CREATE DATABASE 'my\_db'" #添加单引号会报错

* 只需使用cursor.execute()即可，无需提交conn.commit()。否则报错。

## 查询所有数据库SHOW DATABASES

### SQL语句: 查询所有数据库

*SHOW DATABASES*

### Python实例 ：查询所有数据库

* 目标：查询当前服务器上的所有数据库名称
* 实例代码：

import mysql.connector  # 导入模块

# sql = "SHOW DATABASES"  # 定义SQL语句

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

}

try:

    conn = mysql.connector.connect(\*\*config)  # 创建连接

    cursor = conn.cursor()  # 创建游标

    cursor.execute("SHOW DATABASES")  # 执行SQL语句

    result = cursor.fetchall()  # 从游标中获取执行SQL语句返回的结果

    print(result)

except mysql.connector.Error as err:  # 接住异常

    print("Error: " + str(err))

else:

    cursor.close()  # 关闭游标

    conn.close()  # 关闭连接

* 返回结果：



* 细节注意：
* 无需提交事务commit()

## 删除数据库DROP DATABASE

### SQL语句：删除数据库

*DROP DATABASE <db\_name>*

### Python实例：删除数据库

* 目标：删除一个名为my\_db的数据库
* 实例代码：

import mysql.connector  # 导入模块

sql = "SHOW DATABASES"  # 定义SQL语句

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

}

# sql = "CREATE DATABASE my\_db"  # 定义SQL语句

try:

    conn = mysql.connector.connect(\*\*config)  # 创建连接

    cursor = conn.cursor()  # 创建游标

    cursor.execute("DROP DATABASE my\_db")  # 执行SQL语句

    # result = cursor.fetchall()  # 从游标中获取执行SQL语句返回的结果

    # print(result)

except mysql.connector.Error as err:  # 接住异常

    print("Error: " + str(err))

else:

    cursor.close()  # 关闭游标

    conn.close()  # 关闭连接

* 结果验证：

使用sql语句返回所有数据库名



* 常见报错：
* 当删除一个不存在的数据库时，会报错。



* + **解决方法**：利用SQL语句"USE <db\_name>"和try，查看返回的错误代码加以判断，修改流程。

## 修改数据库名称

修改数据库名称的方法，在Innodb引擎和MyISAM引擎是不同。以下是Innodb：

* 方法1：SQL语句

*rename database olddbname to newdbname*

**评价**：官方不推荐，有数据丢失风险。

* 方法2：移库

1.使用新名称创建数据库

2.将数据复制到新库

3.删除旧库

**评价**：数据安全风险最低，但当数据量大时，很耗时。

## 查看指定数据库的信息information\_schema

* 服务器数据库的信息都存储在服务器的information\_schema数据库中。
* 通过SQL语句可以查询指定数据库的信息：

select \* from information\_schema.TABLES where table\_schema = '<db\_name>'

## 检查指定数据库是否存在 Use

* 目标：检查指定数据库是否存在
* 思路：

1.使用SQL语句USE <db\_name>

2.如果报错，检查错误代码

* 实例代码：

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

}

db\_name = 'my\_db'   #待查询的数据库名

try:

    conn = mysql.connector.connect(\*\*config)  # 创建连接

    cursor = conn.cursor()  # 创建游标

    cursor.execute("USE {}".format(db\_name))  # 执行SQL语句

except mysql.connector.Error as err:  # 处理异常

    if err.errno == errorcode.ER\_BAD\_DB\_ERROR:  #检查错误代码

        print("Database {} does not exists.".format(db\_name))

    else:

        print("Error: " + str(err))

else:

    cursor.close()  # 关闭游标

    conn.close()  # 关闭连接

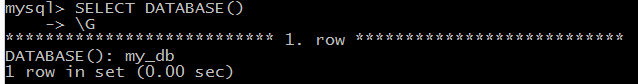
* 返回结果：



## 显示当前数据库名称 select database()

* CMD 命令：

*SELECT DATABASE()*



# 数据表操作

## 创建数据表CREATE TABLE

### SQL语句：创建数据表

*CREATE TABLE <table\_name>*

*(*

*<column\_name1> data\_type(size) constraint\_name,*

*<column\_name2> data\_type(size) constraint\_name,*

*<column\_name3> data\_type(size) constraint\_name,*

*....*

*)*

注意：*<table\_name>, <column\_name>*不需要使用引号括着。

### Python实例：创建数据表

* 目标：

1.创建数据表pet；

2.包含字段name, owner, species, sex, birth, death。

* 实例代码：

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

    'database': 'my\_db'     # 指定数据库

}

sql = "CREATE TABLE pet (\

        name VARCHAR(20), owner VARCHAR(20),\

        species VARCHAR(20), sex CHAR(1), \

        birth DATE, \

        death DATE\

        )"

try:

    conn = mysql.connector.connect(\*\*config)  # 创建连接

    cursor = conn.cursor()  # 创建游标

    cursor.execute(sql)  # 执行SQL语句

except mysql.connector.Error as err:  # 处理异常

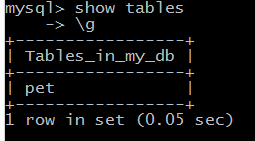
    print("Error: " + str(err))

else:

    cursor.close()  # 关闭游标

    conn.close()  # 关闭连接

* 验证：Show Tables



## 删除数据表DROP TABLE

### SQL语句：删除数据表

*DROP TABLE <table\_name>*

### Python实例：删除数据表

* 目标：删除数据表pet
* 实例代码：

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

    'database': 'my\_db'     # 指定数据库

}

try:

    conn = mysql.connector.connect(\*\*config)  # 创建连接

    cursor = conn.cursor()  # 创建游标

    cursor.execute('DROP TABLE pet')  # 执行SQL语句

except mysql.connector.Error as err:  # 处理异常

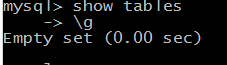
    print("Error: " + str(err))

else:

    cursor.close()  # 关闭游标

    conn.close()  # 关闭连接

* 验证结果：



## 修改数据表Alter Table

### SQL语句：修改数据表

* 添加列：

*ALTER TABLE <table\_name> ADD <column\_name> data\_type(size) constraint\_name*

* 删除列：

*ALTER TABLE <table\_name> DROP COLUMN <column\_name>*

* 改变列的数据类型和约束：

*ALTER TABLE <table\_name> MODIFY COLUMN <column\_name> data\_type(size) constraint\_name*

* 改变列的名、数据类型、约束：

*ALTER TABLE <table\_name> CHANGE <column\_name> data\_type(size) constraint\_name*

### Python实例：修改数据表

* 目标：修改数据表pet

1.增加city列

2.删除owner列

3.修改name列

* 实例代码：

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

    'database': 'my\_db'     # 指定数据库

}

try:

    conn = mysql.connector.connect(\*\*config)  # 创建连接

    cursor = conn.cursor()  # 创建游标

    cursor.execute("ALTER TABLE pet ADD city VARCHAR(20)")      #添加列

    cursor.execute("ALTER TABLE pet DROP COLUMN owner")         #删除列

    cursor.execute("ALTER TABLE pet MODIFY COLUMN name VARCHAR(25)")       #修改列

except mysql.connector.Error as err:  # 处理异常

    print("Error: " + str(err))

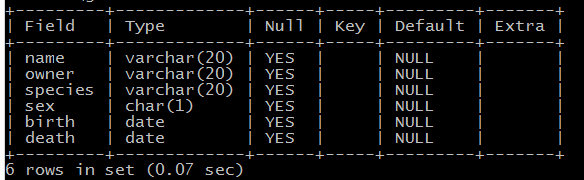
else:

    cursor.close()  # 关闭游标

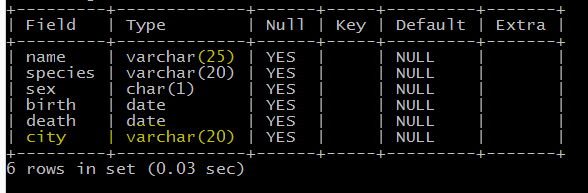
    conn.close()  # 关闭连接

* 验证结果：

修改前：



修改前：



* 常见异常：

如下的组合SQL语句会报错：

sql = "ALTER TABLE pet\

        ADD city VARCHAR(20)\

        ALTER TABLE pet\

        DROP COLUMN owner\

        ALTER TABLE pet\

        MODIFY COLUMN name VARCHAR(25)"

## 显示所有数据表SHOW TABLES

### SQL语句：显示所有数据表

*SHOW TABLES*

### Python实例：

* 目标：获取mysql数据库下的所有表名
* 实例代码：

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

    'database': 'my\_db' # 指定数据库

}

try:

    conn = mysql.connector.connect(\*\*config)  # 创建连接

    cursor = conn.cursor()  # 创建游标

    cursor.execute('SHOW TABLES')  # 执行SQL语句

    result = cursor.fetchall()

    for i in result:

        print(i)

except mysql.connector.Error as err:  # 处理异常

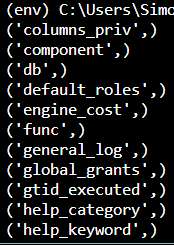
    print("Error: " + str(err))

else:

    cursor.close()  # 关闭游标

    conn.close()  # 关闭连接

* 返回结果：



## 显示指定数据表结构DESCRIBE pet

### SQL语句：显示指定数据表结构

*DESCRIBE <table\_name>*

### Python实例：显示指定数据表结构

* 目标：获取数据表pet的数据表结构
* 实例代码：

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

    'database': 'my\_db'     # 指定数据库

}

try:

    conn = mysql.connector.connect(\*\*config)  # 创建连接

    cursor = conn.cursor()  # 创建游标

    cursor.execute('DESCRIBE pet')  # 执行SQL语句

    result = cursor.fetchall()

    for i in result:

        print(i)

except mysql.connector.Error as err:  # 处理异常

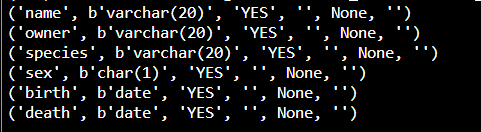
    print("Error: " + str(err))

else:

    cursor.close()  # 关闭游标

    conn.close()  # 关闭连接

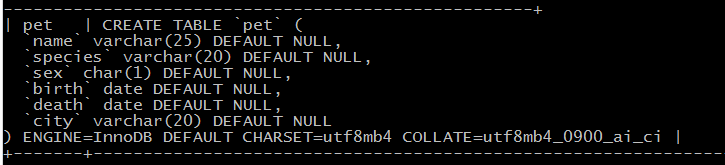
* 返回结果：



## 获取数据表创建语句SHOW CREATE TABLE

### CMD语句

*SHOW CREATE TABLE <tb\_name>*



### Python实例

* 目的：获取数据库pet的创建语句
* 实例代码：

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

    'database': 'my\_db'     # 指定数据库

}

try:

    conn = mysql.connector.connect(\*\*config)  # 创建连接

    cursor = conn.cursor()  # 创建游标

    cursor.execute('SHOW CREATE TABLE pet')  # 执行SQL语句

    result = cursor.fetchall()

    for i in result:

        print(i)

except mysql.connector.Error as err:  # 处理异常

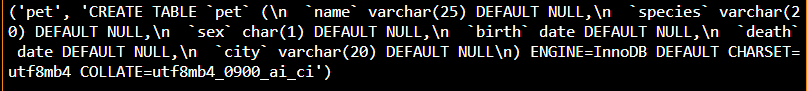
    print("Error: " + str(err))

else:

    cursor.close()  # 关闭游标

    conn.close()  # 关闭连接

* 返回结果：



# 数据基本操作

## 通用示例数据表：

创建通用示例数据表，供下文使用：

sql="CREATE TABLE PERSON \

    (\

    id\_person INT AUTO\_INCREMENT PRIMARY KEY,\

    first\_name VARCHAR(20),\

    last\_name VARCHAR(20)\

    )"

## 插入数据insert into

### SQL语句：插入数据

#### 插入一条数据

*INSERT INTO <table\_name> (column1,column2,column3,...) VALUES (value1,value2,value3,...)*

#### 插入多条数据

*INSERT INTO <table\_name> (column1,column2,column3,...)*

*VALUES*

*(row1\_value1, row1\_value2, row1\_value3,...),*

*(row2\_value1, row2\_value2, row2\_value3,...),*

*...*

### Python实例：插入一条数据

#### 使用SQL语句

* 核心代码：

sql =  "INSERT INTO PERSON (first\_name, last\_name) VALUES('aaa','bbb')"

* 实例代码：插入一条数据

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

    'database': 'my\_db'     # 指定数据库

}

sql =  "INSERT INTO PERSON (first\_name, last\_name) VALUES('aaa','bbb')"

try:

    conn = mysql.connector.connect(\*\*config)  # 创建连接

    cursor = conn.cursor()  # 创建游标

    cursor.execute(sql)  # 执行SQL语句

    conn.commit()       #提交事务

except mysql.connector.Error as err:  # 处理异常

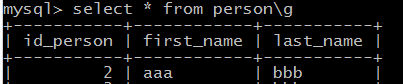
    print("Error: " + str(err))

else:

    cursor.close()  # 关闭游标

    conn.close()  # 关闭连接

* 返回结果：



#### 使用execute(sql,tuple)

* 核心代码：sql语句+tuple

sql = "INSERT INTO PERSON (first\_name, last\_name)\

        VALUES (%s,%s)"

data\_tuple = ('ttt', 'mmm')

    cursor.execute(sql, data\_tuple)     # 执行SQL语句

    conn.commit()       # 提交事务

* 实例代码：

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

    'database': 'my\_db'     # 指定数据库

}

sql = "INSERT INTO PERSON (first\_name, last\_name) VALUES (%s,%s)"

data\_tuple = ('ttt', 'mmm')

try:

    conn = mysql.connector.connect(\*\*config)        # 创建连接

    cursor = conn.cursor()      # 创建游标

    cursor.execute(sql, data\_tuple)     # 执行SQL语句

    conn.commit()       # 提交事务

except mysql.connector.Error as err:        # 处理异常

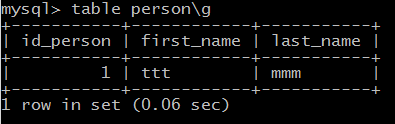
    print("Error: " + str(err))

else:

    cursor.close()      # 关闭游标

    conn.close()        # 关闭连接

* 验证结果：



#### 使用execute(sql,dict)

* 核心代码：sql语句+dict

sql = "INSERT INTO PERSON (first\_name, last\_name) VALUES (%(first\_name)s,%(last\_name)s)"

data\_dict = {'first\_name': 'PPP', 'last\_name': 'LLL'}

    cursor.execute(sql, data\_dict)     # 执行SQL语句

* 实例代码：

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

    'database': 'my\_db'     # 指定数据库

}

sql = "INSERT INTO PERSON (first\_name, last\_name) VALUES (%(first\_name)s,%(last\_name)s)"

data\_dict = {'first\_name': 'PPP', 'last\_name': 'LLL'}

try:

    conn = mysql.connector.connect(\*\*config)        # 创建连接

    cursor = conn.cursor()      # 创建游标

    cursor.execute(sql, data\_dict)     # 执行SQL语句

    conn.commit()       # 提交事务

except mysql.connector.Error as err:        # 处理异常

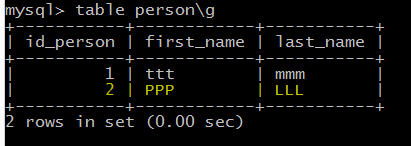
    print("Error: " + str(err))

else:

    cursor.close()      # 关闭游标

    conn.close()        # 关闭连接

* 验证结果：



### Python实例：插入多条数据

#### SQL语句values (),()

* 核心代码：

sql = "INSERT INTO PERSON (first\_name, last\_name) VALUES('111','aaa'),('222','bbb'),('333','ccc')"

* 实例代码：插入一条数据

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

    'database': 'my\_db'     # 指定数据库

}

sql = "INSERT INTO PERSON (first\_name, last\_name) VALUES('111','aaa'),('222','bbb'),('333','ccc')"

try:

    conn = mysql.connector.connect(\*\*config)        # 创建连接

    cursor = conn.cursor()      # 创建游标

    cursor.execute(sql)     # 执行SQL语句

    conn.commit()       # 提交事务

except mysql.connector.Error as err:        # 处理异常

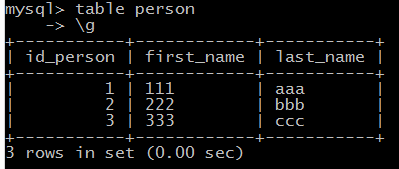
    print("Error: " + str(err))

else:

    cursor.close()      # 关闭游标

    conn.close()        # 关闭连接

* 返回结果：



#### SQL语句values row(), row()

* 核心代码

sql = "INSERT INTO PERSON (first\_name, last\_name) VALUES ROW('444','DDD'),ROW('555','EEE'),ROW('666','FFF')"

* 实例代码：插入一条数据

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

    'database': 'my\_db'     # 指定数据库

}

create\_table = "CREATE TABLE PERSON \

    (\

    id\_person INT AUTO\_INCREMENT PRIMARY KEY,\

    first\_name VARCHAR(20),\

    last\_name VARCHAR(20)\

    )"

sql = "INSERT INTO PERSON (first\_name, last\_name) VALUES ROW('444','DDD'),ROW('555','EEE'),ROW('666','FFF')"

try:

    conn = mysql.connector.connect(\*\*config)        # 创建连接

    cursor = conn.cursor()      # 创建游标

    cursor.execute(sql)     # 执行SQL语句

    conn.commit()       # 提交事务

except mysql.connector.Error as err:        # 处理异常

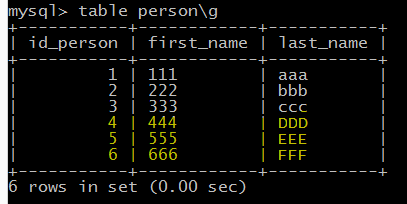
    print("Error: " + str(err))

else:

    cursor.close()      # 关闭游标

    conn.close()        # 关闭连接

* 返回结果：



#### 方法cursor.executemany(%s,tuple\_list)

* 核心代码：SQL语句+元组列表

sql = "INSERT INTO PERSON (first\_name, last\_name)\

        VALUES (%s,%s)"

values\_list=[('qqq','www'),('eee','rrr'),('ttt','yyy')]

    cursor.executemany(sql,values\_list)     # 执行SQL语句

* 实例代码：

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

    'database': 'my\_db'     # 指定数据库

}

sql = "INSERT INTO PERSON (first\_name, last\_name)\

        VALUES (%s,%s)"

values\_list=[('qqq','www'),('eee','rrr'),('ttt','yyy')]

try:

    conn = mysql.connector.connect(\*\*config)        # 创建连接

    cursor = conn.cursor()      # 创建游标

    cursor.executemany(sql,values\_list)     # 执行SQL语句

    conn.commit()       # 提交事务

except mysql.connector.Error as err:        # 处理异常

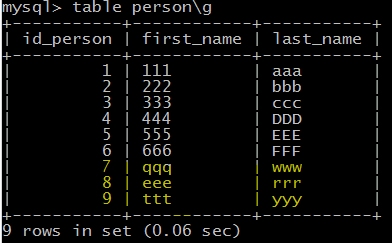
    print("Error: " + str(err))

else:

    cursor.close()      # 关闭游标

    conn.close()        # 关闭连接

* 返回结果：



#### 方法cursor.executemany(%(name)s,dict\_list)

* 核心代码：sql语句+字典列表

sql = "INSERT INTO PERSON (first\_name, last\_name)\

        VALUES (%(first\_name)s,%(last\_name)s)"

values\_list=[{'first\_name':'aaaa','last\_name':'qqqq'},

             {'first\_name':'bbbb','last\_name':'wwww'},

             {'first\_name':'cccc','last\_name':'eeee'}]

    cursor.executemany(sql,values\_list)     # 执行SQL语句

    conn.commit()       # 提交事务

* 实例代码：

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

    'database': 'my\_db'     # 指定数据库

}

sql = "INSERT INTO PERSON (first\_name, last\_name)\

        VALUES (%(first\_name)s,%(last\_name)s)"

values\_list=[{'first\_name':'aaaa','last\_name':'qqqq'},

             {'first\_name':'bbbb','last\_name':'wwww'},

             {'first\_name':'cccc','last\_name':'eeee'}]

try:

    conn = mysql.connector.connect(\*\*config)        # 创建连接

    cursor = conn.cursor()      # 创建游标

    cursor.executemany(sql,values\_list)     # 执行SQL语句

    conn.commit()       # 提交事务

except mysql.connector.Error as err:        # 处理异常

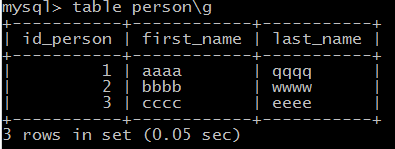
    print("Error: " + str(err))

else:

    cursor.close()      # 关闭游标

    conn.close()        # 关闭连接

* 验证结果：



### 常见异常

* 必须commit()，否则不会执行execute的语句
* 缺少column，如：

INSERT INTO PERSON VALUES('dfdf','frgtg')

* 缺少values，如：

INSERT INTO PERSON (first\_name, last\_name) VALUES()

* column个数与values个数不相等

INSERT INTO PERSON (first\_name, last\_name) VALUES('111')

INSERT INTO PERSON (first\_name) VALUES('111','bbb')

## 查询数据

### SQL语句：

#### 查询指定/所有列select

* 查询指定列

*SELECT <column\_name>,<column\_name> ... FROM <table\_name>*

* 查询所有列

*SELECT \* FROM <table\_name>*

#### 查询所有列table

*table <table\_name>*

### Python实例：select

* 目标：查询指定first\_name列
* 核心代码：

sql = "select first\_name from person"

* 实例代码：

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

    'database': 'my\_db'     # 指定数据库

}

sql = "select first\_name from person"

try:

    conn = mysql.connector.connect(\*\*config)        # 创建连接

    cursor = conn.cursor()      # 创建游标

    cursor.execute(sql)     # 执行SQL语句

    result=cursor.fetchall()

    for i in result:

        print(i)

except mysql.connector.Error as err:        # 处理异常

    print("Error: " + str(err))

else:

    cursor.close()      # 关闭游标

    conn.close()        # 关闭连接

* 返回结果：



### Python实例：table

* 目标：查询所有列
* 核心代码：

sql = "table person"

* 实例代码：

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

    'database': 'my\_db'     # 指定数据库

}

sql = "table person"

try:

    conn = mysql.connector.connect(\*\*config)        # 创建连接

    cursor = conn.cursor()      # 创建游标

    cursor.execute(sql)     # 执行SQL语句

    result=cursor.fetchall()

    for i in result:

        print(i)

except mysql.connector.Error as err:        # 处理异常

    print("Error: " + str(err))

else:

    cursor.close()      # 关闭游标

    conn.close()        # 关闭连接

* 返回结果：



### 常见异常：

* 不需要commit()
* 列名不需要引号：

select 'first\_name' from person

## 删除数据DELETE FROM WHERE

### SQL语句：删除数据

*DELETE FROM <table\_name> WHERE column\_name=value*

### Python实例：删除数据

* 目的：删除first\_name列的值是ttt的行
* 核心代码：

sql = "DELETE FROM person WHERE first\_name='ttt'"

* 实例代码：

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

    'database': 'my\_db'     # 指定数据库

}

sql = "DELETE FROM person WHERE first\_name='ttt'"

try:

    conn = mysql.connector.connect(\*\*config)        # 创建连接

    cursor = conn.cursor()      # 创建游标

    cursor.execute(sql)     # 执行SQL语句

    conn.commit()       # 提交事务

except mysql.connector.Error as err:        # 处理异常

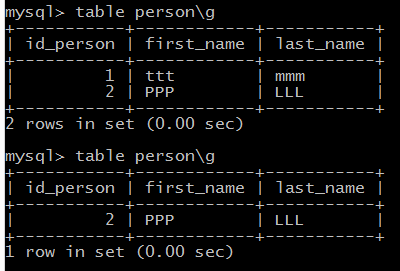
    print("Error: " + str(err))

else:

    cursor.close()      # 关闭游标

    conn.close()        # 关闭连接

* 验证结果：



### 常见异常/错误

* 必须添加where条件，否则会删除所有数据。

## 修改数据update set where

### SQL语句：修改数据

*UPDATE <table\_name>*

*SET column1=value1,column2=value2,...*

*WHERE column=value*

### Python实例：修改数据

* 目的：修改first\_name列的值是PPP的行的last\_name的值
* 核心代码：

sql = "UPDATE person SET last\_name='OOIIU' WHERE first\_name='PPP'"

* 实例代码：

import mysql.connector  # 导入模块

from mysql.connector import errorcode  # 导入异常代码文件

config = {

    'user': 'root',

    'password': 'Simonangel!23fong',

    'host': '127.0.0.1',

    'database': 'my\_db'     # 指定数据库

}

sql = "UPDATE person SET last\_name='OOIIU' WHERE first\_name='PPP'"

try:

    conn = mysql.connector.connect(\*\*config)        # 创建连接

    cursor = conn.cursor()      # 创建游标

    cursor.execute(sql)     # 执行SQL语句

    conn.commit()       # 提交事务

except mysql.connector.Error as err:        # 处理异常

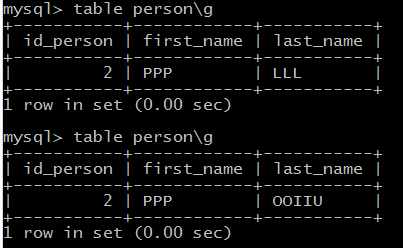
    print("Error: " + str(err))

else:

    cursor.close()      # 关闭游标

    conn.close()        # 关闭连接

* 验证结果：



### 常见异常/错误

* 必须添加where条件，否则会修改所有数据。

# 数据类型

# 约束

# 索引

# END