目录

[重设root密码 2](#_Toc20375)

[Mysql配置文件(my.ini) 2](#_Toc1950)

[数据类型 3](#_Toc29481)

[数值类型 4](#_Toc10070)

[日期和时间类型 4](#_Toc20821)

[字符串类型 4](#_Toc16117)

[逻辑库 5](#_Toc13739)

[数据表 5](#_Toc19318)

[外键约束(FOREIGN KEY) 6](#_Toc12862)

[索引 7](#_Toc923)

[数据操作语言基本操作 7](#_Toc17372)

[高级查询 7](#_Toc21952)

[聚合函数 7](#_Toc6413)

[分组查询 8](#_Toc20840)

[表的内外连接 8](#_Toc6654)

[MySQL的基本操作 8](#_Toc16889)

[MySQL基本函数 9](#_Toc190)

[数学函数 9](#_Toc11391)

[日期函数 10](#_Toc3197)

[字符串函数 11](#_Toc12404)

[条件函数 12](#_Toc28588)

[MySQL数据库的事务机制 13](#_Toc3038)

[用法 13](#_Toc22572)

[mysql事务具有ACID特性 13](#_Toc17728)

[REDO日志 14](#_Toc10742)

[UNDO日志 14](#_Toc21043)

[mysql事务的隔离级别 14](#_Toc2002)

[修改事务的隔离级别 15](#_Toc1952)

[数据导入与导出 15](#_Toc25382)

[加密函数 16](#_Toc18673)

[MySQL与Python的交互 16](#_Toc21221)

[MySQL-Connector模块 16](#_Toc21947)

[游标（cursor） 16](#_Toc13077)

[SQL注入攻击 16](#_Toc20900)

[事务控制 17](#_Toc22933)

[数据库连接池 17](#_Toc16533)

# 重设root密码

创建TXT，定义SQL语句：ALTER USER 'root'@'localhost' IDENTIFIED BY '123456';

打开Power Shell，停止mysql服务：net stop mysql80

输入：mysqld --defaults-file=”数据库my.ini地址” --init-file=”TXT地址” --console

Ctrl+c退出，关闭Power Shell，打开Power Shell，启动mysql服务：net start mysql80

# Mysql配置文件(my.ini)

[client] 客户端配置

[mysql] 客户端配置

[mysqld] 数据库配置

port=3306 -----------------------------------------------------------------------------------

datadir=C:/ProgramData/MySQL/MySQL Server 8.0/Data ----------------------

default\_authentication\_plugin=mysql\_native\_password ------------------------

default-storage-engine=INNODB ------------------------------------------------------

sql-mode="STRICT\_TRANS\_TABLES,NO\_ENGINE\_SUBSTITUTION" -----------

log-output=FILE ----------------------------------------------------------------------------

general-log=0 -------------------------------------------------------------------------------

general\_log\_file="DESKTOP-E8UMQO0.log" --------------------------------------

slow-query-log=1 --------------------------------------------------------------------------

slow\_query\_log\_file="DESKTOP-E8UMQO0-slow.log" --------------------------

long\_query\_time=10 --------------------------------------------------------------------

log-error="DESKTOP-E8UMQO0.err" ------------------------------------------------

log-bin="DESKTOP-E8UMQO0-bin"

server-id=1 ----------------------------------------------------------------------------------

report\_port=3306

lower\_case\_table\_names=1 ------------------------------------------------------------

secure-file-priv="C:/ProgramData/MySQL/MySQL Server 8.0/Uploads" ----

max\_connections=151 -------------------------------------------------------------------

table\_open\_cache=2000

tmp\_table\_size=55M

thread\_cache\_size=10 -------------------------------------------------------------------

myisam\_max\_sort\_file\_size=100G

myisam\_sort\_buffer\_size=100M

key\_buffer\_size=8M

read\_buffer\_size=64K

read\_rnd\_buffer\_size=256K

端口号

数据目录

密码认证插件

默认存储引擎

开启严格模式

用文件记录日志

关闭日志输出

日志文件名

开启慢查询日志

慢查询日志名称

执行多少秒的语句记录在慢查询

错误日志名称

数据库ID

把表明转换成小写

导入导出数据库的地址

最大链接数

线程数量

innodb\_flush\_log\_at\_trx\_commit=1

innodb\_log\_buffer\_size=1M

innodb\_buffer\_pool\_size=8M

innodb\_log\_file\_size=48M

innodb\_thread\_concurrency=9

innodb\_autoextend\_increment=64

innodb\_buffer\_pool\_instances=8

innodb\_concurrency\_tickets=5000

innodb\_old\_blocks\_time=1000

innodb\_open\_files=300

innodb\_stats\_on\_metadata=0

innodb\_file\_per\_table=1

innodb\_checksum\_algorithm=0

back\_log=80

flush\_time=0

join\_buffer\_size=256K

max\_allowed\_packet=4M

max\_connect\_errors=100

open\_files\_limit=4161

sort\_buffer\_size=256K

table\_definition\_cache=1400

binlog\_row\_event\_max\_size=8K

sync\_master\_info=10000

sync\_relay\_log=10000

sync\_relay\_log\_info=10000

# 数据类型

## 数值类型

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类型** | **大小** | **范围(有符号)** | **范围(无符号)** | **用途** |
| TINYINT | 1 byte | (-128，127) | (0，255) | 小整数值 |
| SMALLINT | 2 bytes | (-32 768，32 767) | (0，65 535) | 大整数值 |
| MEDIUMINT | 3 bytes | (-8 388 608，8 388 607) | (0，16 777 215) | 大整数值 |
| INT或INTEGER | 4 bytes | (-2 147 483 648，2 147 483 647) | (0，4 294 967 295) | 大整数值 |
| BIGINT | 8 bytes | (-9,223,372,036,854,775,808，9 223 372 036 854 775 807) | (0，18 446 744 073 709 551 615) | 极大整数值 |
| FLOAT | 4 bytes | (-3.402 823 466 E+38，-1.175 494 351 E-38)，0，(1.175 494 351 E-38，3.402 823 466 351 E+38) | 0，(1.175 494 351 E-38，3.402 823 466 E+38) | 单精度 浮点数值 |
| DOUBLE | 8 bytes | (-1.797 693 134 862 315 7 E+308，-2.225 073 858 507 201 4 E-308)，0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308) | 0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308) | 双精度 浮点数值 |
| DECIMAL | 对DECIMAL(M,D) ，如果M>D，为M+2否则为D+2 | 依赖于M和D的值 | 依赖于M和D的值 | 小数值 |

## 日期和时间类型

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类型** | **大小 ( bytes)** | **范围** | **格式** | **用途** |
| DATE | 3 | 1000-01-01/9999-12-31 | YYYY-MM-DD | 日期值 |
| TIME | 3 | '-838:59:59'/'838:59:59' | HH:MM:SS | 时间值或持续时间 |
| YEAR | 1 | 1901/2155 | YYYY | 年份值 |
| DATETIME | 8 | 1000-01-01 00:00:00/9999-12-31 23:59:59 | YYYY-MM-DD HH:MM:SS | 混合日期和时间值 |
| TIMESTAMP | 4 | 1970-01-01 00:00:00/2038  结束时间是第 **2147483647** 秒，北京时间 **2038-1-19 11:14:07**，格林尼治时间 2038年1月19日 凌晨 03:14:07 | YYYYMMDD HHMMSS | 混合日期和时间值，时间戳 |

## 字符串类型

|  |  |  |
| --- | --- | --- |
| **类型** | **大小** | **用途** |
| CHAR | 0-255 bytes | 定长字符串 |
| VARCHAR | 0-65535 bytes | 变长字符串 |
| TINYBLOB | 0-255 bytes | 不超过 255 个字符的二进制字符串 |
| TINYTEXT | 0-255 bytes | 短文本字符串 |
| BLOB | 0-65 535 bytes | 二进制形式的长文本数据 |
| TEXT | 0-65 535 bytes | 长文本数据 |
| MEDIUMBLOB | 0-16 777 215 bytes | 二进制形式的中等长度文本数据 |
| MEDIUMTEXT | 0-16 777 215 bytes | 中等长度文本数据 |
| LONGBLOB | 0-4 294 967 295 bytes | 二进制形式的极大文本数据 |
| LONGTEXT | 0-4 294 967 295 bytes | 极大文本数据 |

# 逻辑库

创建：CREATE DATABASE 逻辑库名 charset=utf8;

显示：SHOW DATABASE;

删除：DROP DATABASE 逻辑库名;

# 数据表

创建：**CREATE TABLE 数据表(xxx,xxx);**

显示：**SHOW 数据表;**

查看表结构：**DESC 数据表;**

查看创建SQL语句：**SHOW CREATE TABLE 数据表;**

删除：**DROP TABLE 数据表;**

添加字段：**ALTER TABLE 数据表ADD** 字段1 xxx,ADD 字段2 xxx;

修改字段和约束：**ALTER TABLE 数据表MODIFY** 字段1 xxx,MODIFY 字段2 xxx;

修改字段名字：**ALTER TABLE 数据表CHANGE**旧名 新名 xxx;

删除字段：**ALTER TABLE 数据表DROP**字段1 ,DROP字段2 ;

[数据库表字段约束](https://class.imooc.com/lesson/1471" \l "mid=33484)

1. 主键约束(PRIMARY KEY)

2. 唯一性约束(UNIQUE)

3. 非空约束(NOT NULL)

4. 外键约束(FOREIGN KEY)

CREATE TABLE t\_teacher(

id **INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT**,

name VARCHAR(20) NOT NULL,

tel CHAR(11) NOT NULL UNIQUE,

married\_BOOLEAN NOT NULL **DEFAULT FALSE**);

## 外键约束(FOREIGN KEY)

**如果形成外键闭环，将无法删除任何一张表的记录**

CREATE TABLE t\_dept(

deptno INT UNSIGNED PRIMARY KEY ,

dname VARCHAR(20) NOT NULL UNIQUE,

tel CHAR(4) UNIQUE);

CREATE TABLE t\_emp(

empno INT UNSIGNED PRIMARY KEY ,

sex **ENUM("男","女")** NOT NULL,

deptno INT UNSIGNED NOT NULL,

**FOREIGN KEY (deptno) REFERENCES t\_depy(deptno)**);

# 索引

创建索引：**CREATE TABLE 数据表INDEX**字段1 ;

添加索引：**CREATE INDEX** 索引名 **ON** 表名(字段);

**ALTER TABLE** 表名 **ADD INDEX** 索引名(字段);

查看表名：**SHOW INDEX FROM** 表名;

删除索引：**DROP INDEX** 索引名 **ON** 表名;

# 数据操作语言基本操作

数据分页：SELECT 。。。 FROM 。。。 LIMIT 起始 偏移量

排序：SELECT 。。。 FROM 。。。 ORDER BY 列名 [DESC 降序]

去除重复字段：SELECT DISTINCT 字段 FROM 。。。;

条件查询：SELECT 。。。 FROM。。WHERE。。

IFNULL：为NUll替换 IFNULL(字段,0)

DATEDIFF：计算日期偏差 DATEDIFF(NOW()，旧的)第一减第二

IS NULL:是否为空 IS NOT NULL：不为空

BETWEEN AND：范围 BETWEEN 2000 AND 3000

LIKE：模糊查找 LIKE ‘%A%’ / LIKE ‘\_AA’

REGEXP：正则表达式

# 高级查询

## 聚合函数

**AVG([DISTINCT] expr )**：返回 expr 的平均值。 DISTINCT:选项可用于返回 expr 的不同值的平均值。  
**COUNT(expr )**：返回SELECT 语句检索到的行中非NULL 值的数目。若找不到匹配的行，则COUNT() 返回 0  
**MAX([DISTINCT] expr )/MIN([DISTINCT] expr )**：返回 expr 的最小值和最大值。 MIN() 和 MAX() 的取值可以是一个字符串参数；在这些情况下， 它们返回最小或最大字符串值。  
**SUM([DISTINCT] expr )**：返回expr 的总数。 若返回集合中无任何行，则 SUM() 返回NULL 。DISTINCT 关键词可用于 MySQL 5.1 中，求得expr 不同值的总和。 若找不到匹配的行，则SUM() 返回 NULL

## 分组查询

**分组：SELECT 字段1 FROM 。。。GROUP BY字段1**

**GROUP\_CONCAT(expr)：**将字段拼接成一盒字符串

**对结果进行统计：SELECT 字段1 FROM 。。。GROUP BY字段1 WITH ROLLUP**

## 表的内外连接

**SELECT** A.1 B.1 **FROM** A **JOIN** B **on** A.1=B.1

**相同的数据表也可以做表连接**

分类：内连接：只保留符合连接条件的记录

**SELECT** A.1 B.1 **FROM** A **JOIN** B **ON** A.1=B.1 ;

**SELECT** A.1 B.1 **FROM** A **JOIN** B **WHERE** A.1=B.1;

**SELECT** A.1 B.1 **FROM** A **,** B **WHERE** A.1=B.1;

外链接：全保留

**SELECT** A.1 B.1 **FROM** A **LEFT JOIN** B **ON** A.1=B.1 ;

**SELECT** A.1 B.1 **FROM** A **RIGHT JOIN** B **ON** A.1=B.1 ;

# MySQL的基本操作

插入：**INSERT INTO** 表名（字段） **VALUES** （值）；

**INSERT INTO** 表名 **SET** 字段1=值1；

**IGNORE:忽略错误**

修改：**UPDATE** 表名 **SET** 字段1=值1；

**UPDATE** 表1 **JOIN** 表2 **ON** A.1=B.1  **SET** 字段1=值1；

**UPDATE** 表1 **LEFT**/**RIGHT** **JOIN** 表2 **ON** A.1=B.1  **SET** 字段1=值1；

删除：**DELETE** **FROM** 表名

**DELETE** 表1 **FROM** 表1 **JOIN** 表2 **ON** A.1=B.1

**DELETE** 表1 **FROM** 表1 **LEFT**/**RIGHT JOIN** 表2 **ON** A.1=B.1

删除全部：**TRUNCATE TABLE** 表名

# MySQL基本函数

## 数学函数

|  |  |  |
| --- | --- | --- |
| **函数名** | **描述** | **实例** |
| ABS(x) | 返回 x 的绝对值 | SELECT ABS(-1) -- 返回1 |
| CEIL(x) | 返回大于或等于 x 的最小整数 | SELECT CEIL(1.5) -- 返回2 |
| COUNT(expression) | 返回查询的记录总数，expression 参数是一个字段或者 \* 号 | SELECT COUNT(ProductID) AS NumberOfProducts FROM Products; |
| FLOOR(x) | 返回小于或等于 x 的最大整数 | 小于或等于 1.5 的整数：  SELECT FLOOR(1.5) -- 返回1 |
| MOD(x,y) | 返回 x 除以 y 以后的余数 | 5 除于 2 的余数：  SELECT MOD(5,2) -- 1 |
| PI() | 返回圆周率(3.141593） | SELECT PI() --3.141593 |
| POW(x,y) | 返回 x 的 y 次方 | 2 的 3 次方：  SELECT POW(2,3) -- 8 |
| RAND() | 返回 0 到 1 的随机数 | SELECT RAND() --0.93099315644334 |
| ROUND(x) | 返回离 x 最近的整数 | SELECT ROUND(1.23456) --1 |
| SQRT(x) | 返回x的平方根 | 25 的平方根：  SELECT SQRT(25) -- 5 |
| TRUNCATE(x,y) | 返回数值 x 保留到小数点后 y 位的值（与 ROUND 最大的区别是不会进行四舍五入） | SELECT TRUNCATE(1.23456,3) -- 1.234 |

## 日期函数

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **函数名** | **描述** | | **实例** | | |
| NOW() | | | 返回当前日期和时间 | | SELECT NOW()-> 2018-09-19 20:57:43 |
| CURDATE() | | | 返回当前日期 | | SELECT CURDATE();-> 2018-09-19 |
| CURTIME() | | | 返回当前时间 | | SELECT CURTIME();-> 19:59:02 |
| DATE\_FORMAT(d,f) | | | 按表达式 f的要求显示日期 d | | SELECT DATE\_FORMAT('2011-11-11 11:11:11','%Y-%m-%d %r')-> 2011-11-11 11:11:11 AM |
| DATE\_ADD(d，INTERVAL expr type) | | | 计算起始日期 d 加上一个时间段后的日期 | | SELECT ADDDATE('2011-11-11 11:11:11',1)-> 2011-11-12 11:11:11    (默认是天)  SELECT ADDDATE('2011-11-11 11:11:11', INTERVAL 5 MINUTE)-> 2011-11-11 11:16:11 (TYPE的取值与上面那个列出来的函数类似) |
| DATEDIFF(d1,d2) | | | 计算日期 d1->d2 之间相隔的天数 | | SELECT DATEDIFF('2001-01-01','2001-02-02')-> -32 |

## 字符串函数

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **函数** | **描述** | | **实例** | | |
| LOWER(s) | | | 将字符串 s 的所有字母变成小写字母 | | 字符串 RUNOOB 转换为小写：  SELECT LOWER('RUNOOB') -- runoob |
| UPPER(s) | | | 将字符串转换为大写 | | 将字符串 runoob 转换为大写：  SELECT UPPER("runoob"); -- RUNOOB |
| LENGTH(s) | | | 返回字符串 s 的字符数 | |  |
| CONCAT(s1,s2...sn) | | | 字符串 s1,s2 等多个字符串合并为一个字符串 | | 合并多个字符串  SELECT CONCAT("SQL ", "Runoob ", "Gooogle ", "Facebook") AS ConcatenatedString; |
| INSERT(s1,x,len,s2) | | | 字符串 s2 替换 s1 的 x 位置开始长度为 len 的字符串 | | 从字符串第一个位置开始的 6 个字符替换为 runoob：  SELECT INSERT("google.com", 1, 6, "runnob"); -- 输出：runoob.com |
| REPLACE(s,s1,s2) | | | 将字符串 s2 替代字符串 s 中的字符串 s1 | | 将字符串 abc 中的字符 a 替换为字符 x：  SELECT REPLACE('abc','a','x') --xbc |
| SUBSTR(s, start, length) | | | 从字符串 s 的 start 位置截取长度为 length 的子字符串 | | 从字符串 RUNOOB 中的第 2 个位置截取 3个 字符：  SELECT SUBSTR("RUNOOB", 2, 3) AS ExtractString; -- UNO |
| SUBSTRING(s, start, length) | | | 从字符串 s 的 start 位置截取长度为 length 的子字符串 | | 从字符串 RUNOOB 中的第 2 个位置截取 3个 字符：  SELECT SUBSTRING("RUNOOB", 2, 3) AS ExtractString; -- UNO |
| RPAD(s1,len,s2) | | | 在字符串 s1 的结尾处添加字符串 s2，使字符串的长度达到 len | | 将字符串 xx 填充到 abc 字符串的结尾处：  SELECT RPAD('abc',5,'xx') -- abcxx |
| LPAD(s1,len,s2) | | | 在字符串 s1 的开始处填充字符串 s2，使字符串长度达到 len | | 将字符串 xx 填充到 abc 字符串的开始处：  SELECT LPAD('abc',5,'xx') -- xxabc |
| TRIM(s) | | | 去掉字符串 s 开始和结尾处的空格 | | 去掉字符串 RUNOOB 的首尾空格：  SELECT TRIM(' RUNOOB ') AS TrimmedString; |

# 条件函数

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **函数名** | **描述** | | **实例** | | |
| [IFNULL(v1,v2)](https://www.runoob.com/mysql/mysql-func-ifnull.html" \t "https://www.runoob.com/mysql/_blank) | | | 如果 v1 的值不为 NULL，则返回 v1，否则返回 v2。 | | SELECT IFNULL(null,'Hello Word')->Hello Word |
| IF(expr,v1,v2) | | | 如果表达式 expr 成立，返回结果 v1；否则，返回结果 v2。 | | SELECT IF(1 > 0,'正确','错误') ->正确 |
| CASE expression  WHEN condition1 THEN result1  WHEN condition2 THEN result2  ...  WHEN conditionN THEN resultN  ELSE resultEND | | | CASE 表示函数开始，END 表示函数结束。如果 condition1 成立，则返回 result1, 如果 condition2 成立，则返回 result2，当全部不成立则返回 result，而当有一个成立之后，后面的就不执行了。 | | SELECT CASE  　　WHEN 1 > 0  　　THEN '1 > 0'  　　WHEN 2 > 0  　　THEN '2 > 0'  　　ELSE '3 > 0'  　　END->1 > 0 |

# MySQL数据库的事务机制

## 用法

**START TRANSACTION;**

sql语句

**[COMMIT/ROLLBACK]**

## mysql事务具有ACID特性

**原子性（Atomicity）**：事务中的所有操作，要么全部执行，要么都不执行

**一致性（Consistency）**：事务开始和结束后，数据库的完整性不会被破坏

**隔离性（Isolation）**：事务之间互不影响。事务的隔离级别有四种：读未提交、读已提交、可重复读（默认隔离级别）、串行化

**持久性（Durability）**：事务提交后，对数据的修改是永久性的，即使系统故障也不会丢失

日志操作流程：

拷贝数据到UNdo日志，update，delete后写入redo日志，undo日志可以用来恢复数据库，redo日志用来修改数据库。

## REDO日志

事务执行时将需要执行的事务写到REDO文件中。每条SQL进行数据库更新时，先将REDO日志写入日志缓冲区。当客户端执行commit命令时，日志缓冲区内容被刷新到磁盘。

REDO日志对应磁盘上的ib\_logfileN文件，大小默认5M。MYSQL崩溃恢复时会重新执行REDO日志中的操作记录

## UNDO日志

UNDO日志主要用于事务异常时的回滚。复制事务前的数据库内容到UNDO缓冲区，在合适时间将内容刷新到磁盘。

## mysql事务的隔离级别

**读未提交（read uncommited）**：即事务对数据库数据所做的修改，在事务未提交之前，可以被其他事务看到。A事务对数据库数据做的修改，在A事务未提交时，事务B可读到A事务对数据所做的修改。

容易产生的问题：脏读。由于事务B可读到事务A所做的修改，但是A并没有提交，如果事务A执行过程中发生异常回滚，则会导致事务B读到的是错误数据，即脏数据

**读已提交（read commited）**：A、B两个数据同时对数据库中同一条数据做修改，A事务开启，读取数据，此时，B事务也开启，读取数据，并对数据做出修改，然后提交，此时事务A继续执行，执行完后，再次读取数据，发现两次读取的结果不一样，由此产生不可重复读的问题。

**可重复读（repeatable read）**：A事务读取数据库数据，在处理业务的过程中事务B读取数据并修改完成后，事务A再次读取数据时，所得结果和之前读取的数据一致，并且事务A提交后，不会损害数据的一致性。原理是mysql使用了MVCC的版本控制机制，读事物开始时，系统会给当前读事物一个版本号，读事物后续读取的版本号都会<=当前版本号，导致当前的读事物无法读取到数据的最新版本（这里要说明一下，如果在B事物修改数据后A再修改数据并重新读取数据，是可以读取到最新的数据的）。可能产生的问题：幻读。如：事务A开启，读取用户x的age为15，B事务开启age读取用户x的age并加1，此时用户x的age为16，A事务再次读取用户x的信息发现age还是15，A事物将x用户的age+1，提交后，发现x用户的age值为17.从用户的角度来看，数据的一致性并没有被破坏，但A事务可能将读取到的数据15返回给用户，导致幻读。

**串行化（serializable）**：表上的一个事务开启后，在该事务提交之前，其他事务不能对表中的数据做任何修改。该隔离级别会锁表。

## 修改事务的隔离级别

**读未提交（read uncommited） 读已提交（read commited）**

**可重复读（repeatable read） 串行化（serializable）**

SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

## 数据导入与导出

导出表结构和数据：mysqldump -u root -p no-data 逻辑库 > 路径

导出表结构：mysqldump -u root -p 逻辑库 > 路径

导入：

mysql> create database abc ; # 创建数据库

mysql> use abc; # 使用已创建的数据库

mysql> **source** /home/abc/abc.sql # 导入备份数据库

加密函数

二进制：AES\_ENCRYPT(str,key\_str)

转16进制：HEX(AES\_ENCRYPT(str,key\_str))

解密：AES\_DECRYPT(crypt\_str,key\_str)

16进制解密：AES\_DECRYPT(UNHEX(crypt\_str),key\_str)

# MySQL与Python的交互

## MySQL-Connector模块

import mysql.connector  
  
con = mysql.connector.connect(  
 host='localhost', port='3306',  
 user='root', password='123',  
 database='demo'  
)  
con.close()

## 游标（cursor）

用来执行SQL语句，和返回结果集

cursor=con.cursor()  
cursor.execute(SQL语句)

for one in cursor:  
 pass

## SQL注入攻击

可使用预编译防止

cursor.execute(sql,(参数,))

## 事务控制

con.start\_transaction(事务隔离级别)  
con.commit()  
con.rollback()

异常处理

try:  
 con = mysql.connector.connect(  
 host='localhost', port='3306',  
 user='root', password='123',  
 database='demo'  
 )  
 con.start\_transaction()  
 cursor=con.cursor()  
 cursor.execute(sql语句)  
 con.commit()  
except Exception as e:  
 con.rollback()  
 print(e)  
finally:  
 if 'con' in dir():  
 con.close()

# 数据库连接池

pool = mysql.connector.pooling.MySQLConnectionPool(\*\*config,pool\_size=10)  
con=pool.get\_connection()