

虚谷数据库

Python 标准接口 V2.3.3

开发指南

文档版本 01

发布日期 2024-03-15



版权所有 © 2024 成都虚谷伟业科技有限公司。

声明

未经本公司正式书面许可，任何企业和个人不得擅自摘抄、复制、使用本文档中的部分或全部内容，且不得以任何形式进行传播。否则，本公司将保留追究其法律责任的权利。

用户承诺在使用本文档时遵守所有适用的法律法规，并保证不以任何方式从事非法活动。不得利用本文档内容进行任何侵犯他人权益的行为。

商标声明



为成都虚谷伟业科技有限公司的注册商标。

本文档提及的其他商标或注册商标均非本公司所有。

注意事项

您购买的产品或服务应受本公司商业合同和条款的约束，本文档中描述的部分产品或服务可能不在您的购买或使用范围之内。由于产品版本升级或其他原因，本文档内容将不定期进行更新。

除非合同另有约定，本文档仅作为使用指导，所有内容均不构成任何声明或保证。

成都虚谷伟业科技有限公司

地址：四川省成都市锦江区锦盛路 138 号佳霖科创大厦 5 楼 3-14 号

邮编：610023

网址：www.xugudb.com

前言

概述

本文档主要介绍虚谷数据库 Python 驱动接口的主要功能和其简单的外围应用，旨在帮助使用虚谷数据库服务的应用开发人员快速开发有关于数据库交互的接口编程。为使用 Python2.7 以上的语言版本访问虚谷数据库提供技术支持。

读者对象

- 数据库管理员
- 软件工程师
- 技术支持工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 注意	用于传递设备或环境安全警示信息，若不可避免，可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。
 说明	对正文中重点信息的补充说明。“说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修改记录

文档版本	发布日期	修改说明
01	2024-03-15	第一次发布

目录

1	Python 概述	1
1.1	Python 简介	1
1.2	系统结构	1
1.3	开发流程	2
2	快速入手	3
2.1	Python 下载	3
2.2	Python 安装	3
2.2.1	Unix/Linux 平台安装 Python	3
2.2.2	Windows 平台安装 Python	3
2.3	环境变量	4
2.3.1	概述	4
2.3.2	在 Unix/Linux 设置环境变量	4
2.3.3	在 Windows 设置环境变量	5
2.4	Python 驱动接口部署	5
3	数据类型	6
3.1	虚谷数据库数据类型	6
3.2	存储过程输入输出参数类型	9
3.3	参数数据类型	10
4	编程指导	12
4.1	预备工作	12
4.2	建立连接	12
4.2.1	单 IP 建立连接	12
4.2.2	IPS 建立连接	12
4.3	执行语句	13
4.4	version 接口	13
4.5	connect 接口（通过连接对象引用）	13
4.5.1	cursor()	13

4.5.2	close()	14
4.5.3	autocommit()	14
4.5.4	commit()	14
4.5.5	rollback()	15
4.5.6	ping()	16
4.5.7	select_db()	16
4.5.8	begin()	16
4.6	cursor 接口（通过游标对象引用）	17
4.6.1	close()	17
4.6.2	execute()	17
4.6.3	excutebatch()	18
4.6.4	executemany()	19
4.6.5	callproc()	20
4.6.6	callfunc()	21
4.6.7	fetchone()	21
4.6.8	fetchmany()	22
4.6.9	fetchall()	23
4.6.10	nextset()	23
4.6.11	setinputsizes()	24
4.6.12	clearsize()	25
4.6.13	setinputtype()	25
4.6.14	cleartype()	26
4.6.15	getResultcolname()	27
4.6.16	getResultcolseq()	27
4.6.17	getResultcolsize()	28
4.6.18	getResultcolscale()	28
4.6.19	getResultRowid()	29
4.6.20	fetchalldict()	29
4.7	属性设置接口	30
4.7.1	rowcount	30

4.7.2	arraysize	30
4.7.3	description	31
5	Python 驱动接口的常用应用	32
5.1	连接数据库并执行 SQL 语句	32
5.2	获取结果集列属性	33
5.2.1	一次提取多行结果集	33
5.2.2	获取结果集行数	34
5.2.3	执行 SQL 语句的多种方式	35
5.3	大对象的插入与导出	36
5.4	存储过程或存储函数提取结果集	37
5.5	批量插入示例	38
5.6	多结果示例	39
5.7	预处理参数长度和类型示例	39
6	错误码与常见错误定位	41
6.1	错误码详解	41
6.1.1	Error	41
6.1.2	Warning	41
6.1.3	InterfaceError	41
6.1.4	DatabaseError	42
6.1.5	DataError	42
6.1.6	OperationalError	42
6.1.7	InternalError	42
6.1.8	ProgrammingError	43
6.1.9	NotSupportedError	43
6.2	常见错误定位	43
6.2.1	网络不通	43
6.2.2	连接参数错误	43
6.2.3	用户密码连续错误三次	44

1 Python 概述

1.1 Python 简介

Python 提供了高效的高级数据结构，还能简单有效地面向对象编程。Python 语法和动态类型，以及解释型语言的本质，使它成为多数平台上写脚本和快速开发应用的编程语言。

虚谷数据库 Python 驱动接口是为 Python2.7 及以上版本的应用提供数据库访问接口。虚谷数据库 Python 驱动接口基于《Python Database API Specification v2.0》标准编写，接口兼容标准的同时支持如下特征：

- 支持 SQL 标准语法。
- 支持批量插入优化。
- 支持二进制流插入、更新。
- 支持大对象插入及导出。
- 支持多 SQL 语句执行和获取多结果集。
- 支持 TCP/IP 协议。
- 支持 Python 的 datetime 和虚谷数据库时间类型的映射。
- 支持 SSL 加密通讯数据传输。

1.2 系统结构

虚谷数据库 Python 接口提供了统一的客户端访问数据库、获取数据、管理数据的方式，使用如下介绍的核心类完成所有数据库操作。

- 与服务器建立连接。
- 执行 SQL 语句、访问存储过程、使用事务。
- 对结果集的获取。
- 快速获取下一个结果集。
- 输出定位准确的日志内容。

虚谷数据库 Python 驱动接口共有两层：

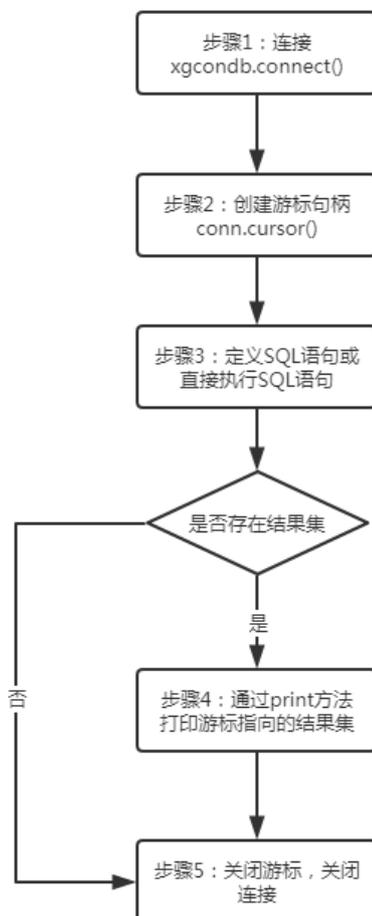
- 上层：用户调用数据的方法，是虚谷数据库 Python 驱动接口提供的类和方法，采用 C 语言编写，面向对象以类对象和其相应的封装方法实现用户操作数据的目的。

- 下层：其内部与数据库交互的部分。

1.3 开发流程

Python 驱动接口使用流程框架如图 1-1 所示。

图 1-1 Python 驱动接口开发流程图



2 快速入手

2.1 Python 下载

访问[Python 官网](#)下载适配的 Python 版本。

📖 说明

发布驱动包支持 Python2.7、Python3.4、Python3.6、Python3.7、Python3.8 与 Python3.9 版本。当前驱动包整合了所有支持的 Python 版本的库文件，切换 Python 版本无需更换驱动包。

更多 Python 详细信息请参见[Python 文档中心](#)，获取 HTML、PDF 和 PostScript 等格式的文
档。

2.2 Python 安装

2.2.1 Unix/Linux 平台安装 Python

操作步骤

1. 打开 WEB 浏览器访问[Python 官方下载中心](#)。
2. 选择适用于 Unix/Linux 的源码压缩包。
3. 下载及解压压缩包。
4. 若需要自定义一些选项修改 Modules/Setup。
5. 执行 ./configure 脚本。
6. Make。
7. make install。

执行以上操作后，Python 会安装在/usr/local/bin 目录中，Python 库安装在/usr/local/lib/pythonXX，XX 为当前使用的 Python 的版本号。

2.2.2 Windows 平台安装 Python

操作步骤

1. 打开 WEB 浏览器访问[Python 官方下载中心](#)。

2. 在下载列表中选择 Windows 平台安装包，包格式为：python-XYZ.msi 文件，XYZ 为安装版本号。

图 2-1 选择版本号



3. 要使用安装程序 python-XYZ.msi，Windows 系统必须支持 Microsoft Installer 2.0。保存安装文件到本地计算机，然后运行它。
4. 下载后，双击下载包，进入 Python 安装向导，安装非常简单，用户只需要使用默认的设置一直点击“下一步”直到安装完成即可。

2.3 环境变量

2.3.1 概述

程序和可执行文件可以存在于多个目录，而这些路径很可能不在操作系统提供可执行文件的搜索路径中。

path(路径) 存储在环境变量中，这是由操作系统维护的一个命名的字符串。这些变量包含可用的命令行解释器和其他程序的信息。

Unix 或 Windows 中路径变量为 PATH（Unix 区分大小写，Windows 不区分大小写）。

在 Mac OS 中，安装程序过程中改变了 Python 的安装路径。若需要在其他目录引用 Python，则必须在 path 中添加 Python 目录。

2.3.2 在 Unix/Linux 设置环境变量

根据自己的 shell 版本来选择其中的某一个选项操作：

- 在 csh shell 输入如下内容，回车。

```
setenv PATH "$PATH:/usr/local/bin/python"
```

- 在 bash shell(Linux) 输入如下内容，回车。

```
export PATH="\$PATH:/usr/local/bin/python"
```

- 在 sh 或者 ksh shell 输入如下内容，回车。

```
PATH="\$PATH:/usr/local/bin/python"
```

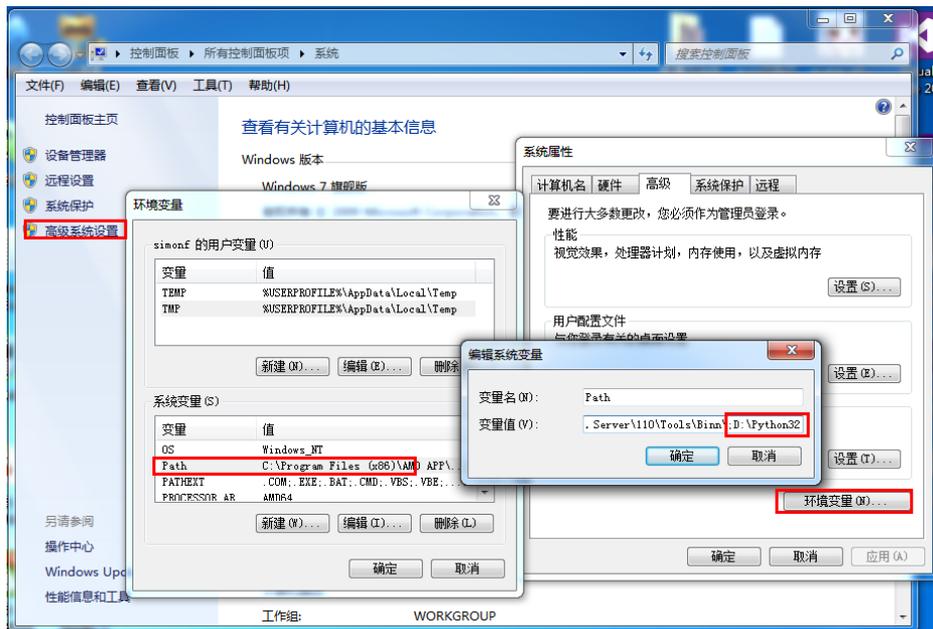
📖 说明

/usr/local/bin/python 是 Python 的安装目录。

2.3.3 在 Windows 设置环境变量

1. 右键点击“计算机”，然后点击“属性”。
2. 点击“高级系统设置”。
3. 选择“系统变量”窗口下的“Path”，双击。
4. 在“Path”行，添加 Python 安装路径即可。
5. 设置成功后，在 CMD 命令行，输入命令“python”，即可获得相关版本信息。

图 2-2 Windows 配置环境变量



2.4 Python 驱动接口部署

Python 驱动接口的安装部署简单快捷，只需获取 Python 驱动接口的扩展包，用户将扩展包放入项目组内，在 Python 代码中 import 即可，Windows 和 Linux 的扩展包名均为“xgcondb”。

3 数据类型

3.1 虚谷数据库数据类型

虚谷数据库数据类型说明和 ODBC 的通用 SQL 数据类型的映射关系如表3-1所示。

表 3-1 虚谷数据库数据类型信息

虚谷数据库数据类型	数据长度 (所占字节)	ODBC SQL 公共数据类型	说明
Char(n)	n 字节, 最大不超过 64K	SQL_CHAR	固定串长度为 n 的字符串
Varchar(n)	n 字节, 最大不超过 64K	SQL_CHAR	最大字符串长度为 n 的可变长度字符串
Binary(n)	n 字节, 最大不超过 64K	SQL_BINARY	固定长度为 n 的二进制数据
Tinyint	1 字节	SQL_TINYINT	精度为 3, 标度为 0 的有符号精确数字, 范围-128– 127
Smallint	2 字节	SQL_SMALLINT	精度为 5, 标度为 0 的有符号精确数字范围 -32768 – 32767
Integer	4 字节	SQL_INTEGER	精度为 10, 标度为 0 的有符号精确数字 C int 的取值
			接下页

虚谷数据库数据类型	数据长度 (所占字节)	ODBC SQL 公共数据类型	说明
Bigint	8 字节	SQL_BIGINT	精度为 19，标度为 0 的有符号精确数字值 int64 的取值范围
Float	4 字节	SQL_FLOAT	浮点数类型
Double	8 字节	SQL_DOUBLE	双精度浮点数字
Bool	1 字节	SQL_TINYINT、 SQLCHAR	布尔型，取值 true, false 或者 'T' , 'F'
Numeric(p,s)	20 字节	SQL_NUMERIC	精度为 p，标度为 s 的有符号精确数字值
Time	4 字节	SQL_TIME	时间数据类型，时分秒字段
Datetime	8 字节	SQL_DATETIME	时间戳数据类型，年月日时分秒字段
Date	4 字节	SQL_DATE	日期数据类型，年月日字段
Time with time zone	6 字节	SQL_CHAR	时间数据类型，时分秒，时区字段
Datetime with time zone	10 字节	SQL_CHAR	时间戳数据类型，年月日时分秒，时区字段
			接下页

虚谷数据库数据类型	数据长度 (所占字节)	ODBC SQL 公共数据类型	说明
Blob	最大不超过 4G	SQL_LONGVARBIN ARY	二进制大对象类型字段
Clob	最大不超过 4G	SQL_LONGVARCH AR	字符大对象的存储字段
Interval year	4 字节	SQL_INTERVAL_Y EAR	年间隔, 即两个日期之间的年数字
Interval month	4 字节	SQL_INTERVAL_M ONTH	月间隔, 即两个日期之间的月数字
Interval day	4 字节	SQL_INTERVAL_D AY	日间隔, 即两个日期之间的日数字
Interval hour	4 字节	SQL_INTERVAL_H OUR	时间间隔, 即为两个日期/时间之间的时数字
Interval minute	4 字节	SQL_INTERVAL_MI NUTE	分间隔, 即为两个日期/时间之间的分数字
Interval second	8 字节	SQL_INTERVAL_S ECOND	秒间隔, 即为两个日期/时间之间的秒数字
Interval day to hour	4 字节	SQL_INTERVAL_D AY_TO_HOUR	日时间间隔, 即为两个日期/时间之间的日时数字
Interval day to minute	4 字节	SQL_INTERVAL_D AY_TO_MINUTE	日时分间隔, 即为两个日期/时间之间的日时分数字
			接下页

虚谷数据库数据类型	数据长度 (所占字节)	ODBC SQL 公共数据类型	说明
Interval day to second	8 字节	SQL_INTERVAL_DAY_TO_SECOND	日时分秒间隔, 即为两个日期/时间之间的日时分秒数字
Interval hour to minute	4 字节	SQL_INTERVAL_HOUR_TO_MINUTE	时分间隔, 即为两个日期/时间之间的时分数字
Interval hour to second	8 字节	SQL_INTERVAL_HOUR_TO_SECOND	时分秒间隔, 即为两个日期/时间之间的时分秒数字
Interval minute to second	8 字节	SQL_INTERVAL_MINUTE_TO_SECOND	分秒间隔, 即为两个日期/时间之间的分秒间隔
Interval year to month	4 字节	SQL_INTERVAL_YEAR_TO_MONTH	年月间隔, 即两个日期之间的年月数字

3.2 存储过程输入输出参数类型

表 3-2 存储过程输入输出参数类型映射表

输入输出类型	整型映射
PARAM_INPUT	1
PARAM_OUTPUT	2
PARAM_INPUTOUTPUT	3
接下页	

输入输出类型	整型映射
PARAM_RETURNVALUE	4

3.3 参数数据类型

表 3-3 参数数据类型映射表

格式符	描述	备注
NULL	XG_C_NULL	0
BOOLEAN	XG_C_BOOL	1
VARCHAR	XG_C_CHAR	2
TINYINT	XG_C_TINYINT	3
SMALLINT	XG_C_SHORT	4
INTEGER	XG_C_INTEGER	5
BIGINT	XG_C_BIGINT	6
FLOAT	XG_C_FLOAT	7
DOUBLE	XG_C_DOUBLE	8
NUMERIC	XG_C_NUMERIC	9
DATE	XG_C_DATE	10
TIME	XG_C_TIME	11
TIME	XG_C_TIME_TZ	12
DATETIME	XG_C_DATETIME	13
DATETIME	XG_C_DATETIME_TZ	14

接下页

格式符	描述	备注
BINARY	XG_C_BINARY	15
BINARY	XG_C_NVARBINARY	18
DATETIME	DATETIME_ASLONG	23
INTERVAL_YEAR_TO_MONTH	XG_C_INTERVAL_YEAR_TO_MONTH	28
INTERVAL_DAY_TO_SECOND	XG_C_INTERVAL_DAY_TO_SECOND	31
CLOB	XG_C_CLOB	41
BLOB	XG_C_BLOB	42
SYS_REFCURSOR	XG_C_REFCUR	58
CHAR	XG_C_NCHAR	62
CHAR	XG_C_CHARN1	63

4 编程指导

4.1 预备工作

对于 Python 脚本，需要让其支持中文输出和 PythonX.X 驱动，创建一个 Python 脚本的时候，在前面写入以下语句：

```
import xgcondb // 导入虚谷数据库 Python 专用驱动
```

导入了“xgcondb”目录后，即可使用虚谷数据库的 Python 专用驱动来进行数据的连接、增删改查等一系列数据库操作。



虚谷数据库 Python 的驱动文件夹 xgcondb 必须和 Python 脚本文件处于同一个目录下。

4.2 建立连接

4.2.1 单 IP 建立连接

在“xgcondb”目录中定义了 connect 方法，通过此函数可以创建连接对象，在调用该函数时，需要一个连接对象接收函数返回值，代码如下：

```
conn=xg condb.connect(host="127.0.0.1",port="5138",database="SYSTEM",user="SYSDBA", password="SYSDBA",charset="GBK")
```

通过导入的 xgcondb 引用其中的 connect 函数，用户需要自己输入相应的 IP 地址、端口号、数据库名、用户名、用户密码和字符集，若连接成功，会返回一个连接对象。

4.2.2 IPS 建立连接

在“xgcondb”目录中定义了 connect 方法，通过此函数可以创建连接对象。在调用该函数时，可以使用多个 IP，IP 与 IP 之间使用逗号分隔。代码如下：

```
conn=xg condb.connect(host="192.168.2.92,192.168.2.93,192.168.2.94",port="5138",database="SYSTEM",user="SYSDBA", password="SYSDBA",charset="GBK")
```

该函数会返回一个连接对象。

4.3 执行语句

当建立了连接，并成功接收到一个连接对象时，可直接通过该连接对象调用函数 `cursor()`，该函数的作用是创建一个游标对象，可以通过该对象调用 `execute()` 函数执行 SQL 语句，代码如下：

```
cur = conn.cursor()           // 创建一个游标对象
cur.execute("create table test2(a int,b boolean,c boolean);") //
    执行无参数SQL语句
```

`execute()` 函数也支持通过对 SQL 语句传参的方式来构建一个完整的 SQL 语句并执行，代码如下：

```
cur = conn.cursor()           // 创建一个游标对象
cur.execute("insert into test2 values(?,?,?);", (234,False,True))
    // 执行有参数SQL语句
```

4.4 version 接口

功能

获取 Python 驱动接口的版本号。

代码示例

```
import xgcondb
print(xgcondb.version())
```

4.5 connect 接口（通过连接对象引用）

4.5.1 cursor()

功能

该函数的作用为创建游标对象，创建方式和创建连接句柄方法一致。

参数

无

返回值

无

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5146", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
```

4.5.2 close()

功能

该函数的作用是关闭当前连接对象。

参数

无

返回值

无

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
conn.close()
```

4.5.3 autocommit()

功能

该函数的作用是设置当前连接是否自动提交事务，默认为自动提交事务。

参数

False 或 True。

返回值

无

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
conn.autocommit(False)           // 设置为非自动提交
conn.autocommit(True)            // 设置为自动提交
conn.close()
```

4.5.4 commit()

功能

该函数在连接提交属性已经设置为非自动提交的前提下使用，当前事务为非自动提交模式，用户在执行 SQL 语句操作后，需主动调用该函数，提交事务，否则事务将处于未提交状态。



事务长时间处于未提交状态将严重影响其它程序访问相关资源。

参数

无

返回值

无

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
conn.autocommit(True)           // 设置为自动提交
conn.autocommit(False)        // 设置为非自动提交
cur.execute("create table test2(a bigint,b boolean,c boolean);")
conn.commit()
conn.close()
```

4.5.5 rollback()

功能

该函数的作用是当前事务回滚，在当前事务处于部分执行但未提交时，可通过调用该函数撤销当前事务的所有操作，一般在其执行 SQL 语句后使用该函数。

📖 说明

当前事务的执行提交操作可分以下两种方法：

- 显式主动调用 `commit()` 方法；
- 执行 DDL 语句。

参数

False 或 True。

返回值

无

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
conn.autocommit(True)           // 设置为非自动提交
conn.autocommit(False)        // 设置为非自动提交
cur.execute("create table test2(a bigint,b boolean,c boolean);")
conn.rollback()
conn.close()
```

4.5.6 ping()

功能

该函数的作用是判断当前连接是否存活，并在连接失效时根据参数进行自动重连。

参数

False（默认）或 True。

返回值

False 或 True。

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
conn.ping()
conn.ping(True)
conn.ping(False)
```

4.5.7 select_db()

功能

切换当前连接对象连接的数据库。

参数

库名。

返回值

无

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
conn.select_db("test_db")
```

4.5.8 begin()

功能

开启一个新的事务，如果存在未提交的事务，则会先提交当前事务再开启新事务。



事务长时间处于未提交状态将严重影响其它程序访问相关资源。

参数

无

返回值

无

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
conn.autocommit(False)    // 设置为非自动提交
conn.begin()
conn.commit()
conn.close()
```

4.6 cursor 接口（通过游标对象引用）

4.6.1 close()

功能

该函数的作用是关闭游标对象。

参数

无

返回值

无

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
cur.close()
conn.close()
```

4.6.2 execute()

功能

该函数是 SQL 语句的执行函数，支持 SQL 语句带参数、不带参数和构造 SQL 语句三种方式。

参数

- SQL 语句。
- 若 SQL 语句存在参数，需要传入参数元组对象。

返回值

无

使用说明

- 该函数支持 DDL、有参数的 DML、无参数的 DML 语句及其他 SQL 语句。
- 执行的 SQL 语句无参数时，则仅需要填写相关 SQL 语句即可。
- 在执行有参数的 insert 语句时，仅支持一组参数（即只插入一条 insert 语句）。
- 绑定参数时，只支持按位置绑定参数。
- 参数输入使用元组或列表的方式。
- 若绑定参数有大对象，将其数据读取存储于 Python 对象中，进行绑定参数；导出大对象也是如此，接口返回对象数据，用户需要创建文件句柄，对象数据写入文件句柄中，详细示例请参见章节5。
- 该函数能执行多结果集 SQL 语句，详细示例请参见章节5。
- 此函数支持大对象的插入与导出，具体示例请参见章节5。

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
#-----execute 执行不带参数的SQL语句-----
cur.execute("create table test2(a int,b boolean,c boolean);")
#-----execute 执行带参数的SQL语句示例1-----
cur.execute("insert into test2 values(?,?,?);", (234,False,True,))
#-----execute 执行带参数的SQL语句示例2-----
sql = "insert into test values(?,?,?,?);"
cur.execute(sql, (3,'xg','xugu','2017-04-27',12.5,12323423.3432))
cur.close()
conn.close()
```

4.6.3 excutebatch()

功能

该函数用于执行批量操作，用法和 execute() 类似。

参数

- SQL 语句。
- 各个参数的集合。

返回值

无

使用说明

- 此函数允许执行批量操作。
- 在传入参数时，须保证各个参数列表（元组）的成员个数一致，否则会因异常退出。

代码示例

```
#!/usr/local/bin/python3
import os
import xgcondb
conn=xgcondb.connect(host="127.0.0.1",port="5138",database="PYTHON3
    ",user="SYSDBA", password="SYSDBA")
cur=conn.cursor()
cur.execute("create table update_tab(d1 int,d2 varchar);")
t_list_1 = []
t_list_2 = []
name = 'Python'
for i in range(5000):
#单次批量执行改变的行数不得超过数据库设置的单个事务最大变更数
    t_list_1.append(i)
    t_list_2.append(name+str(i))
cur.executebatch('insert into update_tab values(?,?);',(t_list_1,
    t_list_2))
cur.execute("select * from update_tab;")
row3=cur.fetchall()
print(row3)
cur.execute("drop table update_tab;")
cur.close()
conn.close()
```

4.6.4 executemany()

功能

该函数是 SQL 语句的批量执行，用法和 execute() 类似，区别是 executemany() 执行 sql 时是用 preparestatement。该函数在执行 SQL 语句前会预处理，效率高于 execute()。

参数

- SQL 语句。
- 若 SQL 语句存在参数，需传入参数元组对象。

返回值

无

使用说明

- 该函数支持接收多组参数。
- 其他使用方式参考 execute 函数。
- 若绑定参数有大对象，将其数据读取存储于 Python 对象中，再进行绑定参数；导出大对象也是如此，接口返回对象数据，用户需要创建文件句柄，对象数据写入文件句柄中，详细示例请参见章节5。
- 该函数不能执行多结果集 SQL 语句。

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
cur.execute("create table test(a int,b varchar,c date,d varchar,
    numeric(8,4) );")
rows=((4, 'xg', 'xugu', '2017-05-26', '12.5', '12323423.3432'), ('4', 'xg'
    , 'xugu', '2017-05-26', '12.5', '12323423.3432'))
cur.executemany(sql, rows)
cur.executemany("select * from dual;")
cur.executemany(sql, (5, 'xg', 'xugu', '2017-07-27'
    , 112.5, 12323423.3432))
cur.close()
conn.close()
```

4.6.5 callproc()

功能

该函数是存储过程的执行函数，支持存储过程带参数（in、out、inout）、存储过程不带参数、存储过程返回结果集。并输出执行之后的相关参数的值。

参数

- 存储过程名。
- 若存储过程需要传递参数，第二个参数应该为参数元组对象。
- 若存储过程需要传递参数，第三个参数应该为参数类型元组对象。

返回值

参数列表对象。

使用说明

- 若存储过程无参数，则在函数内传入存储过程名即可。
- 若存储过程存在参数，需要传入两组参数：第一组为参数数据，第二组为参数的输入输出类型（in、out、inout），两组参数都以元组或列表方式传入，详细信息请参见章节3.2。
- 若存储过程执行后会产生结果集（存储过程内有 select 语句），则需要用户调用 setinputtype 函数提前指定参数的数据类型（XG_C_REFCUR），参数数据类型请参见章节3.3，setinputtype 函数下面会详细解释用法用例，详细示例请参见章节5。
- 函数执行之后会将传入的参数统一输出。

代码示例

```
# 执行无参数存储过程
print(cur.callproc("test_no_parameter"))
# 执行有参数存储过程
print(cur.callproc("test_in", (20,), (1,)))
print(cur.callproc("test_out", ("xugu",), (2,)))
```

```
print(cur.callproc("test_inout", (20, 'xugu'), (3, 2)))
```

4.6.6 callfunc()

功能

该函数是存储函数的执行函数，支持存储函数带参数（in、out、inout）、存储函数不带参数、存储函数返回结果集。并输出执行之后的相关参数的值和函数返回值，与 callproc 函数使用方式相似。

参数

- 存储函数名；
- 若存储函数需要传递参数，第二个参数应该为参数元组对象；
- 若存储函数需要传递参数，第三个参数应该为参数类型元组对象。

返回值

参数列表对象。

使用说明

- 若存储函数无参数，则在函数内传入存储函数名即可；
- 若存储函数存在参数，需要传入两组参数：第一组为参数数据，第二组为参数的输入输出类型（in、out、inout），两组参数都以元组或列表方式传入，详细信息请参见章节3.2。
- 若存储函数执行后会产生结果集（存储函数内有 select 语句），则需要用户调用 setinputtype 函数提前指定参数的数据类型（XG_C_REFCUR），参数数据类型请参见章节3.3，setinputtype 函数下面会详细解释用法用例，详细示例请参见章节5。
- 若使用 setinputtype 函数提前指定参数的数据类型，需要在最后指定存储函数返回的数据类型，否则默认为字符串返回。
- 函数执行之后会将传入的参数和函数返回值统一输出。

代码示例

```
# 执行无参数存储函数  
print(cur.callfunc("test_no_parameter_func"))  
# 执行有参数存储函数  
print(cur.callfunc("test_in_func", (20, ), (1, )))  
print(cur.callfunc("test_out_func", ("xugu", ), (2, )))  
print(cur.callfunc("test_inout_func", (20, 'xugu'), (3, 2)))
```

4.6.7 fetchone()

功能

该函数是用于提取结果集的一行数据，当执行了 `execute()` 的 `select` 操作之后，会有结果存放于结果集，通过该函数即可提取结果集的数据。

参数

无

返回值

结果集一行数据的元组对象。

使用说明

导出大对象，接口返回的是数据，用户需要创建文件句柄，然后将数据写入文件句柄中。

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
cur.executemany("select * from dual;")
cur.fetchone()           //此函数就是提取上面select语句的结果集，每一次提取一行
print (cur.fetchone())  //这样便能打印结果集的信息
cur.close()
conn.close()
```

4.6.8 fetchmany()

功能

该函数也是用于提取结果集的数据，但是与 `fetchone()` 的区别在于，该函数能够从结果集中提取多行数据，用户在使用此函数时，需要先设置提取的行数，然后再使用此函数提取多行数据。

参数

- 若直接通过参数指定需要返回多少行结果集数据，则需要指定，格式：`size=3`；
- 若通过属性 `arraysize` 指定，则无需传递参数。

返回值

结果集多行数据的集合对象（集合对象装的是结果集某一行数据的元组对象），即返回的集合对象内还有多个元组对象，元组对象内装的才是结果集数据。

使用说明

导出大对象，接口返回对象数据，用户需要创建文件句柄，将数据写入文件句柄中。

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="SYSTEM", user="SYSDBA", password="SYSDBA")
```

```
cur = conn.cursor()
cur.executemany("select * from dual;")
cur.arraysize = 3                                // 设置每次提取3行数据
print (cur.fetchmany())                          // 打印结果集中的3行数据

-----

cur.executemany("select * from dual;")
row =cur.fetchmany(size = 3)                     // 提取结果集中的3行数据
print (row)
cur.close()
conn.close()
```

4.6.9 fetchall()

功能

该函数适用于结果集数据较少的情况，作用是一次性提取结果集的所有数据，比 `fetchone()` 和 `fetchmany()` 方便，但是如果结果集数据较多，则要谨慎使用。

参数

无

返回值

结果集剩余数据的集合对象（集合对象装的是结果集每一行数据的元组对象），即返回的集合对象内还有多个元组对象，元组对象内装的才是结果集数据。

使用说明

导出大对象，接口返回的是数据，用户需要创建文件句柄，然后将数据写入文件句柄中。

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
cur.executemany("select * from dual;")
print (cur.fetchall())                          // 提取结果集所有数据并打印
cur.close()
conn.close()
```

4.6.10 nextset()

功能

该方法会使 `cursor` 跳到下一个可用的结果集，并丢弃当前结果集的所有剩余行。

参数

无

返回值

- 存在下一结果集返回 'TRUE' 。
- 不存在下一结果集返回 None。

使用说明

- 若执行的 SQL 语句没有下一个结果集，则输出 None。
- 若存在下一个结果集，则输出 TRUE，并刷新结果集，用户可直接调用 fetchone、fetchmany、fetchall 函数输出结果集信息。

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
cur.executemany("select * from test;select * from test1;")
print (cur.nextset())           // 跳过test表的结果集获取，直接获取test1的结果集
cur.close()
conn.close()
```

4.6.11 setinputsizes()

功能

此方法可以在调用 execute()、executemany()、callproc()、callfunc() 之前使用，用于预定义数据库操作参数所需的内存区域。用户可视需求情况选择使用该方法。

参数

预定义参数长度元组对象。

返回值

无

使用说明

- 此方法用于设置参数缓存空间大小，用户可预先设置参数所需要的内存空间（尤其是字符串）。
- 此方法设置后，会一直影响后面 SQL 函数的执行，直到用户调用 clearsize() 方法清除本次设置，clearsize() 方法使用介绍请参见章节4.6.12。
- 参数设置内存空间，必须与当前插入的参数个数相同，且一一对应。
- 若绑定存储函数，需要在最后加入存储函数返回的参数大小。

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
```

```
#-----设置存储过程的参数长度（三个参数）-----  
cur.setinputsizes((200,4,8))  
print(cur.execproc("test_proc",("xugu",24,65535)))  
cur.clearsize() //清除上面设置的参数空间  
#-----设置存储函数的参数长度（三个参数）-----  
cur.setinputsizes((200,4,8,200))#最后一个参数是函数返回值的预定义大小  
print(cur.execfunc("test_func",("xugu",24,65535)))  
cur.clearsize() //清除上面设置的参数空间  
cur.close()  
conn.close()
```

4.6.12 clearsize()

功能

清除上一次设置的参数缓冲区空间长度。

参数

无

返回值

无

使用说明

该函数与 setinputsizes() 配对使用。

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="SYSTEM", user="SYSDBA", password="SYSDBA")  
cur = conn.cursor()  
cur.setinputsizes((200,4,8))  
cur.clearsize() //清除上面设置的参数空间  
cur.close()  
conn.close()
```

4.6.13 setinputtype()

功能

此方法可在调用 execute()、executemany()、callproc()、callfunc() 之前使用，用于预定义数据库操作参数的数据类型。用户可视需求情况选择使用该方法。

参数

预定义的参数数据类型元组对象。

返回值

无

使用说明

- 此方法是为了设置下面即将执行的带参数的 SQL 语句的各参数类型。
- 用户可不使用此方法，但是若使用了此方法，必须将所有的参数类型全部设置。
- 只有在存储过程或者存储函数内的参数中有 SYS_REFCURSOR 类型，且存储过程和存储函数有结果集产生的情况下，才必须使用此方式设置参数类型（XG_C_REFCUR）。
- 此方法设置后，会一直影响后面 SQL 函数的执行，直到用户调用 cleartype 方法清除本次设置，cleartype 方法使用介绍请参见章节4.6.14。
- 若绑定存储函数，需要在最后加入存储函数返回的参数类型。
- 此函数仅仅对 execproc 和 execfunc 函数起作用，执行普通 SQL 语句无须使用此函数设置参数数据类型。

代码示例

```
import xgcondb
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
#-----设置存储过程的参数类型（三个参数）-----
cur.setinputtype((xgcondb.XG_C_INTEGER,xgcondb.XG_C_CHAR,xgcondb.
    XG_C_REFCUR))
print(cur.execproc("test_proc", (32, "xugu", "ser_cursor")))
print(cur.fetchall())           //打印存储过程产生的结果集数据
cur.cleartype()                 //清除上面设置的参数类型

#-----设置存储函数的参数类型（三个参数）-----
cur.setinputtype((xgcondb.XG_C_INTEGER,xgcondb.XG_C_CHAR,xgcondb.
    XG_C_REFCUR,xgcondb.XG_C_CHAR))#最后一个参数是函数返回值的预定义
    参数类型
print(cur.execfunc("test_func", (32, "xugu", "ser_cursor")))
print(cur.fetchall())           //打印存储过程产生的结果集数据
cur.clearsize()                 //清除上面设置的参数类型
cur.close()
conn.close()
```

4.6.14 cleartype()

功能

清除上一次设置的参数数据类型。

参数

无

返回值

无

使用说明

该函数与 `setinputtype()` 配对使用。

代码示例

```
import xgcondb
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
cur.setinputtype((xgcondb.XG_C_INTEGER,xgcondb.XG_C_CHAR,xgcondb.
    XG_C_REFCUR))
cur.cleartype() //清除上面设置的参数类型
cur.close()
conn.close()
```

4.6.15 getResultcolname()

功能

此方法用于结果集存在的情况下，指定列序号，获取对应列名。

参数

需要获取的指定列的序号。

返回值

指定列的列名。

使用说明

- 此方法必须在结果集存在的情况下才能使用。
- 指定的列序号从 1 开始。

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
cur.execute("select * from dual;")
print(cur.getResultcolname(2))
cur.close()
conn.close()
```

4.6.16 getResultcolseq()

功能

此方法在结果集存在的情况下，指定列名，获取对应列序号。

参数

需要获取的指定列的列名。

返回值

指定列的序号。

使用说明

此方法必须在结果集存在的情况下才能使用。

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
cur.execute("select * from dual;")
print(cur.getResultcolseq("arg"))
cur.close()
conn.close()
```

4.6.17 getResultcolsize()

功能

此方法在结果集存在的情况下，指定列序号，获取对应列的精度。

参数

需要获取的指定列的序号。

返回值

指定列的精度。

使用说明

- 此方法必须在结果集存在的情况下才能使用。
- 指定的列序号从 1 开始。

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
cur.execute("select * from dual;")
print(cur.getResultcolsize(2))
cur.close()
conn.close()
```

4.6.18 getResultcolscale()

功能

此方法在结果集存在的情况下，指定列序号，获取对应列的标度。

参数

需要获取的指定列的序号。

返回值

指定列的标度。

使用说明

- 此方法必须在结果集存在的情况下才能使用。
- 指定的列序号从 1 开始。
- 标度只有 NUMERIC 字段会有值，其他数据类型都是 0。

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
cur.execute("select * from dual;")
print(cur.getResultColscale(2))
cur.close()
conn.close()
```

4.6.19 getResultRowid()

功能

获取最后插入的行的 rowid。

参数

无

返回值

最后插入行的 rowid。

使用说明

此方法必须在插入行数据后使用。

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
cur.execute("insert into test_id(id) values(1);")
print(cur.getResultRowid())
cur.close()
conn.close()
```

4.6.20 fetchalldict()

功能

该函数与 fetchall() 功能一致，区别在于 fetchall() 以元组方式返回行数据，该函数以字典方式返回行数据。

参数

无

返回值

结果集剩余数据的集合对象（集合对象中包含结果集每一行数据的字典对象）。

使用说明

导出大对象，接口返回的是数据，用户需要创建文件句柄，然后将数据写入文件句柄中。

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
cur.executemany("select * from dual;")
print (cur.fetchalldict())           //提取结果集所有数据并打印
cur.close()
conn.close()
```

4.7 属性设置接口

4.7.1 rowcount

功能

该函数的作用是获取结果集的行数，在执行完一句 select 的 SQL 语句时，可通过该函数确定输出结果集共有多少行数据。

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
cur.executemany("select * from dual;")
print (cur.rowcount)                 //打印select结果集的行数
cur.close()
conn.close()
```

4.7.2 arraysze

功能

该函数作用是指定需要提取的结果集行数，与 fetchmany() 联用，实现一次提取结果集的多行数据。

代码示例 [language=xgsql]

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
cur.executemany("select * from dual;")
cur.arraysize = 3
data = cur.fetchmany()
print (data)
cur.close()
conn.close()
```

4.7.3 description

功能

该函数作用是获取结果集的列的属性，用户执行 SQL 查询语句后，存在结果集，则会返回结果集中每一列的属性，如果不存在结果集，则会返回 None，如：

Select: 查询之后会返回一个结果集，调用 description() 并打印会输出表每一列的列名和数据类型大小。

代码示例

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
cur.execute("CREATE TABLE TAB_DESCRIPTION_TEST1(A INT, B INT, C
    VARCHAR, D DATETIME, E NUMBER(4,2))") // 创建一个表
print (cur.description)
cur.execute("SELECT * FROM TAB_DESCRIPTION_TEST1;") // 查询表中数据
print (cur.description) // 打印结果集中列的属性
cur.close()
conn.close()
```

5 Python 驱动接口的常用应用

5.1 连接数据库并执行 SQL 语句

```
#!/usr/bin/python3
import xgcondb

conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
try:
    # execute()
    print("execute()")
    cur.execute("CREATE TABLE TAB_FETCHONE_TEST1(A INT, B INT,
C VARCHAR, D DATETIME, E NUMBER(4,2))")
    cur.execute("INSERT INTO TAB_FETCHONE_TEST1 VALUES
(1001,2001,'XUGU1','2019-01-01',23.54);")
    cur.execute("INSERT INTO TAB_FETCHONE_TEST1 VALUES
(1002,2002,'XUGU2','2019-01-02',2.354);")
    cur.execute("INSERT INTO TAB_FETCHONE_TEST1 VALUES
(1003,2003,'XUGU3','2019-01-03',35.4);")
    cur.execute("INSERT INTO TAB_FETCHONE_TEST1 VALUES
(1004,2004,'XUGU4','2019-01-04',54.00);")
    cur.execute("SELECT * FROM TAB_FETCHONE_TEST1;")
    row = cur.fetchone()
    while row is not None:
        print(row)
        row = cur.fetchone()
    print()
    cur.execute("SELECT * FROM TAB_FETCHONE_TEST1;")
    row = cur.fetchone()
    print(row)
    print(cur.fetchone())
    print(cur.fetchone())

    # executemany()
    print("executemany()")
    cur.executemany("SELECT * FROM TAB_FETCHONE_TEST1;")
    row = cur.fetchone()
    while row is not None:
        print(row)
        row = cur.fetchone()
    print()
    cur.executemany("SELECT * FROM TAB_FETCHONE_TEST1;")
    row = cur.fetchone()
    print(row)
    print(cur.fetchone())
```

5.2 获取结果集列属性

```
#!/usr/bin/python3
import xgcondb

conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA")
cur = conn.cursor()
# execute
cur.execute("CREATE TABLE TAB_DESCRIPTION_TEST1(A INT, B INT, C
    VARCHAR, D DATETIME, E NUMBER(4,2))")
print(cur.description)
cur.execute("SELECT * FROM TAB_DESCRIPTION_TEST1;")
print(cur.description)
print("")
cur.execute("SELECT 1 AS NUM FROM DUAL;")
print(cur.description)
cur.execute("SELECT 'xugu' AS NUM FROM DUAL;")
print(cur.description)
cur.execute("DROP TABLE TAB_DESCRIPTION_TEST1;")
print("")
# executemany
cur.execute("CREATE TABLE TAB_DESCRIPTION_TEST2(A INT, B INT, C
    VARCHAR, D DATETIME, E NUMBER(4,2))")
print(cur.description)
cur.execute("SELECT * FROM TAB_DESCRIPTION_TEST2;")
print(cur.description)

cur.execute("SELECT 1 AS NUM FROM DUAL;")
print(cur.description)
cur.execute("DROP TABLE TAB_DESCRIPTION_TEST2;")

# exception test
cur.execute("CREATE TABLE TAB_DESCRIPTION_TEST3(A INT, B INT, C
    VARCHAR, D DATETIME, E NUMBER(4,2))")
cur.execute("SELECT * FROM TAB_DESCRIPTION_TEST3;")
cur.execute("CREATE TABLE TAB_DESCRIPTION_TEST4(F INT, G INT, H
    VARCHAR) ")
try:
    cur.execute("CREATE TABLE TAB_DESCRIPTION_TEST4(F INT, G INT, H
        VARCHAR) ")
except Exception as e:
    print(e)
cur.execute("SELECT * FROM TAB_DESCRIPTION_TEST4;")
print(cur.description)
cur.execute("DROP TABLE TAB_DESCRIPTION_TEST3;")
cur.execute("DROP TABLE TAB_DESCRIPTION_TEST4;")
```

5.2.1 一次提取多行结果集

```
#!/usr/bin/python3
import xgcondb
```

```
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
SYSTEM", user="SYSDBA", password="SYSDBA")

cur = conn.cursor()
try:
    #cur.execute("SELECT * FROM TAB_FETCHMANY_TEST1;")
    #execute()
    print("execute()")
    #cur.execute("CREATE TABLE TAB_FETCHMANY_TEST1(A INT, B INT
    , C VARCHAR, D DATETIME, E NUMBER(4,2))")
    cur.execute("INSERT INTO TAB_FETCHMANY_TEST1 VALUES
(1001,2001,'XUGU1','2010-01-01',23.54);")
    cur.execute("INSERT INTO TAB_FETCHMANY_TEST1 VALUES
(1002,2002,'XUGU2','2011-01-02',2.354);")
    cur.execute("INSERT INTO TAB_FETCHMANY_TEST1 VALUES
(1003,2003,'XUGU3','2012-01-03',35.4);")
    cur.execute("INSERT INTO TAB_FETCHMANY_TEST1 VALUES
(1004,2004,'XUGU4','2013-01-04',54.00);")
    cur.execute("INSERT INTO TAB_FETCHMANY_TEST1 VALUES
(1005,2005,'XUGU5','2014-01-05',54.10);")
    cur.execute("INSERT INTO TAB_FETCHMANY_TEST1 VALUES
(1006,2006,'XUGU6','2015-01-06',54.20);")
    cur.execute("INSERT INTO TAB_FETCHMANY_TEST1 VALUES
(1007,2007,'XUGU7','2016-01-07',54.30);")
    cur.execute("SELECT * FROM TAB_FETCHMANY_TEST1;")
    print(cur.arraysize)
    #cur.arraysize=8
    row = cur.fetchmany()
    print(row)

    print("-----")
    cur.execute("SELECT * FROM TAB_FETCHMANY_TEST1;")
    row = cur.fetchmany(size = 2)
    for ro in row:
        print(ro)
    print("+++++++")
    cur.arraysize = 3
    print(cur.fetchmany())
    print("*****")
    cur.arraysize = 5
    print(cur.fetchmany())
    for ro in row:
        print(ro)
    print(cur.fetchone())
    cur.execute("DROP TABLE TAB_FETCHMANY_TEST1;")
except Exception as e:
    cur.execute("DROP TABLE TAB_FETCHMANY_TEST1;")
```

5.2.2 获取结果集行数

```
#!/usr/bin/python3
import xgcondb

conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
SYSTEM", user="SYSDBA", password="SYSDBA")
```

```
cur = conn.cursor()

##DDL
# create
cur.execute("CREATE TABLE TAB_ROWCOUNT_TEST(A INT,B VARCHAR, C
    DATETIME);")
print("CREATE:", cur.rowcount)

##DML
# INSERT
cur.execute("INSERT INTO TAB_ROWCOUNT_TEST VALUES(1, 'TA
    ', '2019-01-29');")
print("INSERT ONE:", cur.rowcount)
cur.execute("INSERT INTO TAB_ROWCOUNT_TEST VALUES(2, 'TB
    ', '2019-02-28');")
print("INSERT TWO:", cur.rowcount)
cur.execute("INSERT INTO TAB_ROWCOUNT_TESTVALUES(3, 'TC
    ', '2019-03-29')(2, 'TB', '2019-02-28');")
print("INSERT THREE:", cur.rowcount)

# UPDATE
cur.execute("UPDATE TAB_ROWCOUNT_TEST SET B='NN' WHERE A=2;")
print("UPDATE:", cur.rowcount)
# SELECT
cur.execute("SELECT * FROM TAB_ROWCOUNT_TEST;")
print("SELECT:", cur.rowcount)
# DELETE
cur.execute("DELETE FROM TAB_ROWCOUNT_TEST WHERE A=3;")
print("DELETE:", cur.rowcount)
# drop
cur.execute("DROP TABLE TAB_ROWCOUNT_TEST;")
print("DROP:", cur.rowcount)
conn.close()
```

5.2.3 执行 SQL 语句的多种方式

```
#!/usr/bin/python3
import xgcondXugub

# 建立数据库连接, 客户端与数据库连接字符集默认GBK, 可以设置 UTF8
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA", charset="GBK")
# 创建连接的游标
cur = conn.cursor()
# 设置非自动提交, 默认自动提交
conn.autocommit(False)
# 直接执行SQL语句的方式
# DDL语句
cur.execute("create table test2(a bigint,b boolean,c boolean,d
    varchar);")
# DML语句
cur.execute("insert into test2 values(?,?,?,?);", (234,False,True,
    None))
conn.commit();
cur.execute("select * from test2;")
```

```
try:
    rows = cur.fetchall()
    print(rows)
except Exception as e:
    print(e)
cur.execute("create table test(a int, b varchar(10),c char(100),d
    datetime,e double,f numeric(18,4));")
cur.execute("insert into test values(1,'xg', '
    xugu','2017-05-26',12.5,12323423.3432);")
conn.commit();
cur.execute("insert into test values(%i,'%s','%s','%s',%f,%f);"
    % (2,'xg','xugu','2017-05-26',12.5,12323423.3432))
conn.commit();
# execute() 使用参数形式执行SQL语句
# 备注：目前支持的Python的数据类型包括：整形，字符型，浮点型，
    boolean，None
sql = "insert into test values(?,?,?,?);"
# 第二个参数表示数据库行数据，建议使用Tuple类型，支持List，不支持
    dict
cur.execute(sql, (3,'xg','xugu','2017-04-27',12.5,12323423.3432))
cur.execute(sql, [3,'xg','xugu','2017-06-27',12.5,12323423.3432])
cur.execute(sql, ((3,'xg','xugu','2019-01-10',13.5,12323423.3432)
    , (3,'xg','xugu','2019-01-10',13.6,12323423.3432)))
cur.execute(sql, [(3,'xg','xugu','2019-02-10',14.6,12323423.3432)
    , (3,'xg','xugu','2019-02-10',14.6,12323423.3432)])
cur.execute(sql, [[3,'xg','xugu','2019-03-10'
    ,15.7,12323423.3432],[3,'xg','xugu','2019-03-10'
    ,15.6,12323423.3432]])
cur.execute(sql, (([3,'xg','xugu','2019-04-10'
    ,16.8,12323423.3432],[3,'xg','xugu','2019-04-10'
    ,16.6,12323423.3432])))
conn.commit();
# executemany() 批量执行，备注：使用方式和支持类型参考execute()，区
    别：executemany()执行insert时是用preparestatement
rows = ((4,'xg','xugu','2017-05-26','12.5','12323423.3432'),('4','
    xg','xugu','2017-05-26','12.5','12323423.3432'))
cur.executemany(sql,rows)
cur.executemany("select * from dual;")
cur.executemany(sql, (5,'xg','xugu','2017-07-27'
    ,112.5,12323423.3432))
conn.commit()
cur.execute("select * from test;")
try:
    rows = cur.fetchall()
    print(rows)
except Exception as e:
    print(e)
cur.execute("drop table test;")
cur.execute("drop table test2;")
conn.close()
```

5.3 大对象的插入与导出

```
#!/usr/bin/python3
import xgcondb

# 建立数据库连接, 客户端与数据库连接字符集默认GBK, 可以设置 UTF8
conn = xgcondb.connect(host="127.0.0.1", port="5138", database="
    SYSTEM", user="SYSDBA", password="SYSDBA", charset="GBK")
# 创建连接的游标
cur = conn.cursor()
cur.execute('create table t_lob(B BLOB, C CLOB);')
#-----大对象插入-----
clob_fp = open("./clob_test.txt", "r")
blob_fp = open("./blob_test.jpg", "rb")
clob_buf = clob_fp.read()
blob_buf = blob_fp.read()
cur.execute('insert into t_lob values(?,?);', (blob_buf, clob_buf))

#-----大对象导出-----
cur.execute('select * from t_lob;')
row = cur.fetchone()
clob_fd = open("./clob_select.txt", "w+")
blob_fd = open("./blob_select.jpg", "wb+")
blob_fd.write(row[0])
clob_fd.write(row[1])
cur.close()
conn.close()
```

5.4 存储过程或存储函数提取结果集

```
#!/usr/local/bin/python3
import xgcondb
conn=xgcondb.connect(host="127.0.0.1",port="5138",database="PYTHON3
    ",user="SYSDBA", password="SYSDBA");
cur=conn.cursor();
cur.execute('create table test(arg1 int, arg2 varchar);')
#-----存储过程提取结果集-----
cur.execute('''CREATE or replace procedure pro_test(col1 int,col2
    OUT SYS_REFCURSOR) as \
declare \
par1 int; \
str2 varchar; \
begin \
par1:=col1; \
for i in 1..par1 loop \
    str2:='insert into test values('||i||','||par2||');' \
    execute immediate str2; \
end loop; \
OPEN col2 FOR SELECT * FROM test; \
end;''');
cur.setInputtype((xgcondb.XG_C_INTEGER,xgcondb.XG_C_REFCUR))
cur.setInputsizes((4,10))
print(cur.callproc('pro_test',(100,'refcur'),(1,2)))
row = cur.fetchall()
```

```
for i in row:
    print(i)
cur.clearsize()
cur.cleartype()
cur.execute('drop procedure pro_test;')

#----- 存储函数提取结果集 -----
cur.execute('''CREATE or replace function fun_test(coll int,col2
    OUT SYS_REFCURSOR) return varchar as \
declare \
par1 int; \
str2 varchar; \
begin \
par1:=coll; \
for i in 1..par1 loop \
    str2:='insert into test values('||i||','||par2||');'; \
    execute immediate str2; \
end loop; \
OPEN col2 FOR SELECT * FROM test; \
return str2; \
end;''');
cur.setInputtype((xgcondb.XG_C_INTEGER,xgcondb.XG_C_REFCUR,xgcondb.
    XG_C_CHAR))
cur.setInputsizes((4,10,200))
print(cur.callfunc('fun_test',(100,'refcur'),(1,2)))
row = cur.fetchall()
for i in row:
    print(i)
cur.clearsize()
cur.close()
conn.close()
```

5.5 批量插入示例

```
#!/usr/local/bin/python3
import os
import xgcondb
conn=xgcondb.connect(host="127.0.0.1",port="5138",database="PYTHON3
    ",user="SYSDBA", password="SYSDBA")
cur=conn.cursor()
cur.execute("create table update_tab(d1 int,d2 varchar);")
t_list_1 = []
t_list_2 = []
name = 'Python'
for i in range(5000):
    #单次批量执行改变的行数不得超过数据库设置的单个事务最大变更数
    t_list_1.append(i)
    t_list_2.append(name+str(i))
cur.executebatch('insert into update_tab values(?,?);',(t_list_1,
    t_list_2))
cur.execute("select * from update_tab;")
row3=cur.fetchall()
print(row3)
```

```
cur.execute("drop table update_tab;")
cur.close()
conn.close()
```

5.6 多结果示例

```
#!/usr/local/bin/python3
import xgcondb
conn=xgcondb.connect(host="127.0.0.1",port="5138",database="PYTHON3
    ",user="SYSDBA", password="SYSDBA");
cur=conn.cursor();
cur.execute('select * from update_tab; select count(*) from
    update_tab;')
print(cur.fetchall())    #获取第一个结果集的所有数据
print(cur.nextset())     #切换至下一个结果集
print(cur.fetchone())   #获取第二个结果集的数据
cur.close()
conn.close()
```

5.7 预处理参数长度和类型示例

```
#!/usr/local/bin/python3
import xgcondb
conn=xgcondb.connect(host="127.0.0.1",port="5138",database="PYTHON3
    ",user="SYSDBA", password="SYSDBA");
cur=conn.cursor();
cur.execute('create table test(arg1 int, arg2 varchar);')
#-----存储过程提取结果集-----
cur.execute('''CREATE or replace procedure pro_test(coll int,col2
    OUT SYS_REFCURSOR) as \
declare \
par1 int; \
str2 varchar; \
begin \
par1:=coll; \
for i in 1..par1 loop \
    str2:='insert into test values('||i||','||par2||');'; \
    execute immediate str2; \
end loop; \
OPEN col2 FOR SELECT * FROM test; \
end;''');
cur.setInputtype((xgcondb.XG_C_INTEGER,xgcondb.XG_C_REFCUR))
cur.setInputsizes((4,10))
print(cur.callproc('pro_test',(100,'refcur'),(1,2)))
row = cur.fetchall()
for i in row:
    print(i)
cur.clearsize()
cur.cleartype()
cur.execute('drop procedure pro_test;')
```

```
#-----存储函数提取结果集-----
cur.execute('''CREATE or replace function fun_test(col1 int,col2
  OUT SYS_REFCURSOR) return varchar as \
declare \
par1 int; \
str2 varchar; \
begin \
par1:=col1; \
for i in 1..par1 loop \
    str2:='insert into test values('||i||','||par2||');' \
    execute immediate str2; \
end loop; \
OPEN col2 FOR SELECT * FROM test; \
return str2; \
end;''');
cur.setInputtype((xgcondb.XG_C_INTEGER,xgcondb.XG_C_REFCUR,xgcondb.
  XG_C_CHAR))
cur.setInputsizes((4,10,200))
print(cur.callfunc('fun_test',(100,'refcur'),(1,2)))
row = cur.fetchall()
for i in row:
    print(i)
cur.clearsize()
cur.close()
conn.close()
```

6 错误码与常见错误定位

6.1 错误码详解

6.1.1 Error

错误码	函数返回值	错误描述	分析与建议
Error	-1	通用型错误	一般性错误，如 SQL 语句有错等，错误信息描述段内部会有较为详细的内容，注意排查

6.1.2 Warning

错误码	函数返回值	错误描述	分析与建议
Warning	-1	一般警告，如数据字段右侧截断等	这类错误一般不影响执行的过程，但对数据的精度等有影响，需根据错误信息详细排查

6.1.3 InterfaceError

错误码	函数返回值	错误描述	分析与建议
InterfaceError	-1	数据库接口错误	接口调用错误，请参考对照 demo 示例文件的使用情况

6.1.4 DatabaseError

错误码	函数返回值	错误描述	分析与建议
DatabaseError	-1	数据库错误	请检验环境设置，驱动名称引用等

6.1.5 DataError

错误码	函数返回值	错误描述	分析与建议
DataError	-1	数据错误	可能为 SQL 语句中数据列类型和列值的类型不匹配，类型转化失败或超界，需对照表定义进行排查

6.1.6 OperationalError

错误码	函数返回值	错误描述	分析与建议
OperationalError	-1	与编程人员无关的数据库错误：连接丢失、内存分配错误、事务处理错误等	建议先使用较为独立的数据库环境对程序流程进行测试，发现修改问题

6.1.7 InternalError

错误码	函数返回值	错误描述	分析与建议
InternalError	-1	数据库遇到内部错误，例如，游标无效、事务不同步	建议先使用较为独立的数据库环境对程序流程进行测试，发现修改问题

6.1.8 ProgrammingError

错误码	函数返回值	错误描述	分析与建议
ProgrammingError	-1	未找到表、SQL 语句中的语法错误、指定参数的数量错误等	一般这类错误需要对照 SQL 语句和数据库端的表的情况，判断 SQL 和表的结构是否吻合、参数绑定是否合理

6.1.9 NotSupportedError

错误码	函数返回值	错误描述	分析与建议
NotSupportedError	-1	调用的 API 组件并不支持	某些可选调用方法不在支持的范畴，请根据需要选用其它的接口

6.2 常见错误定位

6.2.1 网络不通

错误定位

通过本机 PING 虚谷数据库服务器的 IP 地址。

解决方法

修改本机或者虚谷数据库 IP 地址，确保两个 IP 地址在同一网段或者中间有 VPN 连接。

6.2.2 连接参数错误

错误定位

报错 Connection information is not correct，检查连接字符串是否与虚谷数据库相应参数（IP 地址、端口号、数据库名、用户名、用户密码等）匹配。

解决方法

修改代码中的连接字符串，确保其中的 IP 地址、端口、数据库名、用户名和用户密码正确。

6.2.3 用户密码连续错误三次

错误定位

判断是否存在连续三次输入用户名密码错误的情况。

解决方法

等待 IP 冻结时间（3 分钟）结束，然后输入正确的用户名密码。



成都虚谷伟业科技有限公司

联系电话：400-8886236

官方网站：www.xugudb.com