DbtPy: 高级原生扩展模块

API描述表

DbtPy.active

bool DbtPy.active(IFXConnection connection)

描述

- 检查IFXConnection是否处于活动状态

参数

- connection - 有效的IFXConnection连接

返回值

- True - 资源处于活动状态
- False - 资源处于未激活状态

示例
参考代码：test_002_active.py

```
    bool = DbtPy.active(conn)
    if bool :
      print("连接状态正常")
    else :
      print("连接状态异常")
```

DbtPy.autocommit

mixed DbtPy.autocommit (IFXConnection connection [, bool value])

描述

- 返回并设置指定IFXConnection的AUTOCOMMIT行为

参数

- connection - 有效的IFXConnection连接
- value - 以下参数之一:
    - SQL_AUTOCOMMIT_OFF
    - SQL_AUTOCOMMIT_ON

返回值

- 非指定参数value时:
    - 0 - AUTOCOMMIT值是关闭
    - 1 - AUTOCOMMIT值是打开
- 指定参数value时:
    - True - AUTOCOMMIT值设置成功
    - False - AUTOCOMMIT值未设置成功

示例
参考代码：test_006_autocommit.py

```
print("默认自动提交状态: " + str(commit))
commit = DbtPy.autocommit(conn, DbtPy.SQL_AUTOCOMMIT_OFF)
if commit :
    print("设置自动提交状态为OFF成功")
else :
    print("设置自动提交状态为OFF失败")
commit = DbtPy.autocommit(conn)
print("获取当前自动提交状态: " + str(commit))
```

DbtPy.bind_param

bool DbtPy.bind_param (IFXStatement stmt, int parameter-number, string variable [, int parameter-type [, int data-type [, int precision [, int scale [, int size]]]]])

描述

- 将Python变量绑定到DbtPy.prepare()返回的IFXStatement中的SQL语句参数。与简单地将变量作为可选输入元组的一部分传递给DbtPy.execute()相比，该函数为参数类型、数据类型、精度和参数扩展提供了更多的控制。

参数

- stmt - 从DbtPy.prepare()返回的预编译语句
- parameter-number - 从序号1开始的参数
- variable - 绑定到parameter-number指定的参数的Python变量
- parameter-type - 指定参数输入、输出的常量:
    - SQL_PARAM_INPUT - 仅输入参数

    - SQL_PARAM_OUTPUT - 仅输出参数
    - SQL_PARAM_INPUT_OUTPUT - 输入及输出参数
    - PARAM_FILE - 数据存储在变量中指定的文件名中，而不是变量本身中。这可以用来避免在内存中存储大量的LOB数据。

- data-type - 指定Python变量应该绑定为的SQL数据类型常量，仅接受以下值：

    - SQL_BINARY
    - SQL_CHAR
    - SQL_DOUBLE
    - SQL_LONG

- precision - 变量的精度

- scale - 变量的精度

返回值

- True - 绑定变量成功
- None - 绑定变量不成功

示例
参考代码： test_026_prepare.py

```
    statment = DbtPy.prepare(conn, "select tabid, tabname from systables where
tabid = ? and tabname = ?")
    DbtPy.bind_param(statment, 1, 1, DbtPy.SQL_PARAM_INPUT, DbtPy.SQL_INTEGER)
    DbtPy.bind_param(statment, 2, 'systables', DbtPy.SQL_PARAM_INPUT,
DbtPy.SQL_VARCHAR)
    DbtPy.execute(statment)
    result = DbtPy.fetch_tuple(statment)
    # print(result)
    print("Tabid   : {}" . format(str(result[0])))
    print("Tabname : {}" . format(str(result[1])))
```

DbtPy.callproc

(IFXStatement [, ...]) DbtPy.callproc(IFXConnection connection, string procname [, parameters])

描述

- 调用存储过程。存储过程调用的每个参数(IN/INPUT/OUT)为parameters一个元组。返回的IFXStatement，包含结果集和输入参数的修改副本。IN参数保持不变，INOUT/OUT参数可能会被更新。存储过程可能会0个或者多个结果集。使用DbtPy.fetch_assoc()，DbtPy.fetch_both()，或者DbtPy.fetch_tuple()从IFXStatement获取一行tuple/dict。或者，使用DbtPy.fetch row()将结果集指针移动到下一行，并使用DbtPy.result()一次获取一列。
- 示例参考： test_146_CallSPINAndOUTParams.py，test_148_CallSPDiffBindPattern_01.py 或者test_52949_TestSPIntVarcharXml.py。

参数

- connection - 有效的IFXConnection
- procname - 有效的存储过程名称
- parameters - 包含存储过程所需的任意多个参数的元组

返回值

- 成功，包含IFXStatement对象的元组，后跟传递给过程的参数(如果有的话)
- 不成功，值为none

示例
参考代码： test_025_callproc.py

```
    statment, ptabid, ptabname = DbtPy.callproc(conn, "proc1", (ptabid,
ptabname,))
    result = DbtPy.fetch_both(statment)
    if result :
        print(result)
        print(result[0])
```

注意：看起来out/inout参数存在问题

DbtPy.client_info

object DbtPy.client_info (IFXConnection connection)

描述

- 返回关于客户的只读对象信息

参数

- connection - 有效的IFXConnection

返回值

- 成功，对象包括如下信息：
    - DATA_SOURCE_NAME - 用于创建到数据库的当前连接的数据源名称(DSN)
    - DRIVER_NAME - 实现调用级别接口(CLI)规范的库的名称
    - DRIVER_ODBC_VER - ODBC驱动程序的版本。这将返回一个字符串“MM.mm”，其中MM是主要版本，mm是次要版本。
    - DRIVER_VER - 客户端的版本，以字符串“MM.mm.uuuu”的形式。MM是主版本，mm是次版本，uuuu是更新版本。例如，“08.02.0001”表示主版本8，次版本2，更新1。
    - ODBC_SQL_CONFORMANCE - 客户机支持三种级别的ODBC SQL语法
        - MINIMAL -支持最小ODBC SQL语法
        - CORE - 支持核心ODBC SQL语法
        - EXTENDED - 支持扩展ODBC SQL语法

    - ODBC_VER - ODBC驱动程序管理器支持的ODBC版本。以字符串“MM.mm.rrrr”的形式。MM是主版本，mm是次版本，rrrr是更新版本。客户端总是返回"03.01.0000"

- 不成功，False

示例
参考代码：test_004_client_info.py

```
client = DbtPy.client_info(conn)
if client :
  print("客户端信息: ")
  print("DATA_SOURCE_NAME: \"%s\"" % client.DATA_SOURCE_NAME)
  print("DRIVER_NAME: \"%s\"" % client.DRIVER_NAME)
  print("DRIVER_VER: \"%s\"" % client.DRIVER_VER)
  print("DRIVER_ODBC_VER: \"%s\"" % client.DRIVER_ODBC_VER)
  print("ODBC_VER: \"%s\"" % client.ODBC_VER)
  print("ODBC_SQL_CONFORMANCE: \"%s\"" % client.ODBC_SQL_CONFORMANCE)

  DbtPy.close(conn)
else:
  print("客户端信息错误.")
```

DbtPy.close

bool DbtPy.close (IFXConnection connection)

描述

- 关闭指定的IFXConnection

参数

- connection - 有效的IFXConnection

返回值

- True为成功，False为失败

示例
参考代码：test_003_close.py

```
bool = DbtPy.close(conn)
if bool :
  print("关闭数据库连接正常")
else :
  print("关闭数据库连接异常")
```

DbtPy.column_privileges

IFXStatement DbtPy.column_privileges (IFXConnection connection [, string qualifier [, string schema [, string table-name [, string column-name]]]])

描述

- 返回一个结果集，包含列出表的列和相关权限。

参数

- connection - 有效的IFXConnection
- schema - 包含表的模式。如果要匹配所有模式，请传递None或空字符串。
- table-name - 表或视图的名称。如果要匹配数据库中的所有表，请传递None或空字符串。
- column-name - 列的名称。如果要匹配表中的所有列，请传递None或空字符串。

返回值

- IFXStatement其结果集包含以下列的行
  - TABLE_CAT - catalog的名称。如果数据库没有catalog，则为Nono。
  - TABLE_SCHEM - schema的名称。
  - TABLE_NAME - 表或者视图的名称。
  - COLUMN_NAME - 字段名称。
  - GRANTOR - 授予权限者。
  - GRANTEE - 被授权者。
  - PRIVILEGE - 字段权限。
  - IS_GRANTABLE - 是否允许授权给他人。

示例
参考代码：test_012_column_privileges.py

```
    statement = DbtPy.column_privileges(conn, None, None, 'tab1', None)
    dataRecord = DbtPy.fetch_assoc(statement)
    while dataRecord:
        print("TABLE_CAT   : {}" .format(dataRecord['TABLE_CAT']))
        print("TABLE_SCHEM : {}" .format(dataRecord['TABLE_SCHEM']))
        print("TABLE_NAME  : {}" .format(dataRecord['TABLE_NAME']))
        print("COLUMN_NAME : {}" .format(dataRecord['COLUMN_NAME']))
        print("GRANTOR     : {}" .format(dataRecord['GRANTOR']))
        print("GRANTEE     : {}" .format(dataRecord['GRANTEE']))
        print("PRIVILEGE   : {}" .format(dataRecord['PRIVILEGE']))
        print("IS_GRANTABLE: {}" .format(dataRecord['IS_GRANTABLE']))
        # print(dataRecord)
        dataRecord = DbtPy.fetch_assoc(statment)
```

DbtPy.columns

IFXStatement DbtPy.columns (IFXConnection connection [, string qualifier [, string schema [, string table-name [, string column-name]]]])

描述

- 返回列出表的列和相关元数据的结果集。

参数

- connection - 有效的IFXConnection
- schema - 包含表的模式。如果要匹配所有模式，请传递'%'。
- table-name - 表或视图的名称。如果要匹配数据库中的所有表，请传递None或空字符串。
- column-name - 列的名称。如果要匹配表中的所有列，请传递None或空字符串。

返回值

- IFXStatement其结果集包含以下列的行
  - TABLE_CAT - catalog的名称。如果数据库没有catalog，则为Nono。
  - TABLE_SCHEM - schema的名称。
  - TABLE_NAME - 表或者视图的名称。
  - COLUMN_NAME - 字段名称。
  - DATA_TYPE - 表示为整数值的列的SQL数据类型。
  - TYPE_NAME - 表示列的数据类型的字符串。
  - COLUMN_SIZE - 表示列大小的整数值。
  - BUFFER_LENGTH - 存储来自此列的数据所需的最大字节数。
  - DECIMAL_DIGITS - 列的刻度，如果不适用刻度，则为None。
  - NUM_PREC_RADIX - 整数值，可以是10(表示精确的数字数据类型)，2(表示近似的数字数据类型)，或者None(表示基数不适用的数据类型)。
  - NULLABLE - 整数值，表示列是否可为空。
  - REMARKS - 字段描述信息。
  - COLUMN_DEF - 字段默认值。
  - SQL_DATA_TYPE - 列的SQL数据类型。
  - SQL_DATETIME_SUB - 表示datetime子类型代码的整数值，对于不适用此值的SQL数据类型，则为None。
  - CHAR_OCTET_LENGTH - 字符数据类型列的最大字节长度，对于单字节字符集数据，该长度与列大小匹配，对于非字符数据类型，该长度为None。
  - ORDINAL_POSITION - 列在表中的索引位置（以1开始）。
  - IS_NULLABLE - 字符串值中的“YES”表示该列可为空，“NO”表示该列不可为空。

示例

参考代码：test_011_columns.py

```
    statment = DbtPy.columns(conn, None, None, "tab1", "col1")
    dataRecord = DbtPy.fetch_assoc(statment)
    while dataRecord:
        print("TABLE_CAT   : {}" .format(dataRecord['TABLE_CAT']))
        print("TABLE_SCHEM : {}" .format(dataRecord['TABLE_SCHEM']))
        print("TABLE_NAME  : {}" .format(dataRecord['TABLE_NAME']))
        print("COLUMN_NAME : {}" .format(dataRecord['COLUMN_NAME']))
        print("DATA_TYPE   : {}" .format(dataRecord['DATA_TYPE']))
        print("TYPE_NAME   : {}" .format(dataRecord['TYPE_NAME']))
        print("COLUMN_SIZE : {}" .format(dataRecord['COLUMN_SIZE']))
        print("BUFFER_LENGTH: {}" .format(dataRecord['BUFFER_LENGTH']))
        print("DECIMAL_DIGITS: {}" .format(dataRecord['DECIMAL_DIGITS']))
```

```
        print("NUM_PREC_RADIX: {}" .format(dataRecord['NUM_PREC_RADIX']))
        print("NULLABLE    : {}" .format(dataRecord['NULLABLE']))
        print("REMARKS     : {}" .format(dataRecord['REMARKS']))
        print("COLUMN_DEF  : {}" .format(dataRecord['COLUMN_DEF']))
        print("SQL_DATA_TYPE: {}" .format(dataRecord['SQL_DATA_TYPE']))
        print("SQL_DATETIME_SUB: {}" .format(dataRecord['SQL_DATETIME_SUB']))
        print("CHAR_OCTET_LENGTH: {}" .format(dataRecord['CHAR_OCTET_LENGTH']))
        print("ORDINAL_POSITION: {}" .format(dataRecord['ORDINAL_POSITION']))
        print("IS_NULLABLE : {}" .format(dataRecord['IS_NULLABLE']))

        # print(dataRecord)
        dataRecord = DbtPy.fetch_assoc(statment)
```

DbtPy.commit

bool DbtPy.commit (IFXConnection connection)

描述

- 在指定的IFXConnection上提交一个正在进行的事务，并开始一个新的事务。
- Python应用程序通常默认为自动提交模式，所以没有必要使用DbtPy.commit()，除非在IFXConnection中关闭了自动提交。
- 注意: 如果指定的IFXConnection是一个持久连接，则所有使用该持久连接的应用程序正在进行的所有事务都将被提交。因此，不建议在需要事务的应用程序中使用持久连接。

参数

- connection - 有效的IFXConnection

返回值

- True为成功，False为失败

示例
参考代码：test_007_commit.py

```
commit = DbtPy.autocommit(conn, DbtPy.SQL_AUTOCOMMIT_OFF)
if commit :
    print("设置自动提交状态为OFF成功")
else :
    print("设置自动提交状态为OFF失败")
commit = DbtPy.commit(conn)
if commit :
    print("提交成功")
else :
    print("提交失败")
```

DbtPy.conn_error

string DbtPy.conn_error ([IFXConnection connection])

描述

- 如果没有传递任何参数，则返回表示上一次数据库连接失败原因的SQLSTATE。
- 当传递一个由DbtPy.connect()返回的有效IFXConnection时，返回SQLSTATE，表示上次使用IFXConnection的操作失败的原因。

参数

- connection - 有效的IFXConnection

返回值

- 返回包含SQLSTATE值的字符串，如果没有错误，则返回空字符串。

示例
参考代码： test_033_conn_error.py

```
    try :
        # 错误信息，可修改
        conn = DbtPy.connect(connStr,connUser,connUser)
    except :
        print("连接失败, sqlstate = {}" . format(DbtPy.conn_error()))
```

DbtPy.conn_errormsg

string DbtPy.conn_errormsg ([IFXConnection connection])

描述

- 如果没有传递任何参数，则返回一个字符串，其中包含SQLCODE和表示上次数据库连接尝试失败的错误消息。
- 当传递一个由DbtPy.connect()返回的有效的IFXConnection时，返回一个字符串，其中包含SQLCODE和错误消息，表示上次使用IFXConnection的操作失败的原因。

参数

- connection - 有效的IFXConnection

返回值

- 返回包含SQLCODE和错误消息的字符串，如果没有错误，则返回空字符串。

示例
参考代码： test_034_conn_errormsg.py

```
    try :
        # 错误信息，可修改
        conn = DbtPy.connect(connStr,connUser,connUser)
    except :
        print("连接失败, sqlstate = {}" . format(DbtPy.conn_errormsg()))
```

DbtPy.connect

IFXConnection DbtPy.connect(string ConnectionString, string user, string password [, dict options [, constant replace_quoted_literal])

描述

- 创建一个新到GBase 8s数据库的连接

参数

- ConnectionString以下格式的连接字符串，"PROTOCOL=onsoctcp;HOST=192.168.0.100;SERVICE=9088;SERVER=gbase01;DATABASE=testdb;DB_LOCALE=zh_CN.utf8;CLIENT_LOCALE=zh_CN.utf8"，参考GBase 8s数据库连接参数，其中常用的参数如下：
  - PROTOCOL - 协议类型，常用有onsoctcp, olsoctcp等。
  - HOST - 数据库服务器的主机名或者IP地址。
  - SERVICE - 数据库服务器的侦听端口。
  - SERVER - 数据库服务名称/实例名称。
  - DATABASE - 数据库名称。
  - DB_LOCALE - 数据库服务使用的字符集。
  - CLIENT_LOCALE - 数据库客户端使用的字符集。
- user - 连接到数据库的用户名称。
- password - 用户的密码。

返回值

- 成功，返回IFXConnection对象
- 不成功，None

示例
参考代码：test_001_connect.py

```
try:
  conn = DbtPy.connect(connStr,connUser,connPass)
except:
  print("连接失败，测试成功")
  return -1
print("连接成功，测试失败")
```

DbtPy.cursor_type

int DbtPy.cursor_type (IFXStatement stmt)

描述

- 返回IFXStatement使用的游标类型。使用此参数可确定您使用的是只向前游标还是可滚动游标。

参数

- stmt - 有效的IFXStatement.

返回值

- 以下值之一:
  - SQL_CURSOR_FORWARD_ONLY
  - SQL_CURSOR_KEYSET_DRIVEN
  - SQL_CURSOR_DYNAMIC
  - SQL_CURSOR_STATIC

示例
参考代码： test_027_cursor_type.py

```
    resultSet = DbtPy.exec_immediate(conn, sqlStatement, stmtOption)

    #get the cursor type
    cursorType = DbtPy.cursor_type(resultSet)
    print("Cursor Type : {}" . format(str(cursorType)))
```

DbtPy.exec_immediate

stmt_handle DbtPy.exec_immediate(IFXConnection connection, string statement [, dict options])

描述

- 准备并执行一条SQL语句。
- 如果您计划使用不同的参数重复地执行相同的SQL语句，请考虑调用DbtPy.prepare()和DbtPy.execute()，以使数据库服务器能够复用其访问计划，并提高数据库访问的效率。
- 如果您计划将Python变量插入到SQL语句中，请理解这是一种更常见的安全性暴露。考虑调用DbtPy.prepare()来为输入值准备带有参数标记的SQL语句。然后可以调用DbtPy.execute()传入输入值并避免SQL注入攻击。

参数

- connection - 有效的IFXConnection
- statement - 一个SQL语句。语句不能包含任何参数标记。
- options -包含语句选项的dict。
  - SQL_ATTR_CURSOR_TYPE - 将游标类型设置为以下类型之一(并非所有数据库都支持)
    - SQL_CURSOR_FORWARD_ONLY
    - SQL_CURSOR_KEYSET_DRIVEN
    - SQL_CURSOR_DYNAMIC
    - SQL_CURSOR_STATIC

返回值

- 如果成功发出SQL语句，则返回一个stmt句柄资源；如果数据库执行SQL语句失败，则返回False。

示例

参考代码：test_013_exec_immediate.py

```
    statment = DbtPy.exec_immediate(conn, "select tabid,tabname from systables
where tabid = 1")
    dataRecord = DbtPy.fetch_assoc(statment)
    while dataRecord:
        print("Tabid   : {}" . format(dataRecord['tabid']))
        print("Tabname : {}" . format(dataRecord['tabname']))
        # print(dataRecord)
        dataRecord = DbtPy.fetch_assoc(statment)

    DbtPy.close(conn)
```

DbtPy.execute

bool DbtPy.execute (IFXStatement stmt [, tuple parameters])

描述

- DbtPy.execute()执行由DbtPy.prepare()准备的SQL语句。如果SQL语句返回一个结果集，例如，返回一个或多个结果集的SELECT语句，则可以使用DbtPy.fetch_assoc()，DbtPy.fetch_both() 或 DbtPy.fetch_tuple()从stmt资源中检索作为元组或字典的行。
- 或者，您可以使用DbtPy.fetch row()将结果集指针移动到下一行，并使用DbtPy.result()从该行每次获取一列。有关使用DbtPy.prepare()和DbtPy.execute()而不是使用DbtPy.exec_immediate()的优点的简短讨论，请参阅DbtPy.prepare()。要执行存储过程，参考DbtPy.callproc()。

参数

- stmt - 从DbtPy.prepare()返回的预编译语句。
- parameters - 匹配预置语句中包含的任何参数标记的输入参数元组。

返回值

- 成功返回Ture，失败返回False

示例

参考代码： test_026_prepare.py

```
    statment = DbtPy.prepare(conn, "select tabid, tabname from systables where
tabid = ? and tabname = ?")
    DbtPy.bind_param(statment, 1, 1, DbtPy.SQL_PARAM_INPUT, DbtPy.SQL_INTEGER)
    DbtPy.bind_param(statment, 2, 'systables', DbtPy.SQL_PARAM_INPUT,
DbtPy.SQL_VARCHAR)
    DbtPy.execute(statment)
    result = DbtPy.fetch_tuple(statment)
    # print(result)
    print("Tabid   : {}" . format(str(result[0])))
    print("Tabname : {}" . format(str(result[1])))
```

DbtPy.execute_many

mixed DbtPy.execute_many(IFXStatement stmt, tuple seq_of_parameters)

描述

- 对在参数序列找到的所有参数序列或映射执行由DbtPy.prepare()准备的SQL语句。

参数

- stmt - 从DbtPy.prepare()返回的预编译语句。
- seq_of_parameters - 一个元组的元组，每个元组都包含与预备语句中包含的参数标记相匹配的输入参数。

返回值

- 成功，返回（insert/update/delete）操作的行数
- 不成功，返回None。使用DbtPy.num_rows()查询（inserted/updated/deleted）操作的行数。

示例
参考代码： test_028_execute_many.py

```
prepare = DbtPy.prepare(conn, "insert into tabmany values(?,?)")
param = ((1,'test001'),
(2,'test002'),
(3,'test003'),
(4,'test004'))

result = DbtPy.execute_many(prepare, param)
row_count = DbtPy.num_rows(prepare)
print("插入行数 (num_rows())             :  {} ".format(row_count))
print("插入行数 (execute_many()返回值)   :  {} ".format(result))
```

注意：execute_many()返回的行数不可信，仅可认为是成功。

DbtPy.fetch_tuple

tuple DbtPy.fetch_tuple (IFXStatement stmt [, int row_number])

描述

- 返回按列位置索引的元组，表示结果集中的行。

参数

- stmt - 包含结果集的有效stmt资源。
- row_number - 从结果集中请求特定的索引为1开始的行。如果结果集中使用只向前游标，传递此参数将导致警告。

返回值

- 返回一个元组，其中包含所有结果集的列值为选定的行，如果没有指定行号则为下一行。
- 如果没有行结果集,或者请求的行结果集的行号不存在，返回False。

示例
参考代码：test_015_fetch_tuple.py

```
    statment = DbtPy.exec_immediate(conn, "select tabid,tabname from systables
where tabid = 1")
    dataRecord = DbtPy.fetch_tuple(statment)
    while dataRecord:
        print("Tabid   : {}" . format(dataRecord[0]))
        print("Tabname : {}" . format(dataRecord[1]))
        # print(dataRecord)
        dataRecord = DbtPy.fetch_tuple(statment)
```

DbtPy.fetch_assoc

dict DbtPy.fetch_assoc (IFXStatement stmt [, int row_number])

描述

- 返回以列名为索引的dict，表示结果集中的行。

参数

- stmt - 包含结果集的有效stmt资源。
- row_number - 从结果集中请求特定的索引为1开始的行。如果结果集中使用只向前游标，传递此参数将导致警告。

返回值

- 返回一个元组，其中包含所有结果集的列值为选定的行，如果没有指定行号则为下一行。
- 如果没有行结果集,或者请求的行结果集的行号不存在，返回False。

示例
参考代码：test_014_fetch_assoc.py

```
    statment = DbtPy.exec_immediate(conn, "select tabid,tabname from systables
where tabid = 1")
    dataRecord = DbtPy.fetch_assoc(statment)
    while dataRecord:
        print("Tabid   : {}" . format(dataRecord['tabid']))
        print("Tabname : {}" . format(dataRecord['tabname']))
        # print(dataRecord)
        dataRecord = DbtPy.fetch_assoc(statment)

    DbtPy.close(conn)
```

DbtPy.fetch_both

dict DbtPy.fetch_both (IFXStatement stmt [, int row_number])

描述

- 返回按列名称和位置索引的字典，表示结果集中的行。

参数

- stmt - 包含结果集的有效stmt资源。
- row_number - 从结果集中请求特定的索引为1开始的行。如果结果集中使用只向前游标，传递此参数将导致警告。

返回值

- 返回一个dict，其中包含所有按列名索引的列值，如果未指定行号，则按0索引的列号索引选定行或下一行。
- 如果结果集中没有剩下的行，或者行号请求的行在结果集中不存在，则返回False。

示例
参考代码：test_016_fetch_both.py

```
    statment = DbtPy.exec_immediate(conn, "select tabid,tabname from systables
where tabid = 1")
    dataRecord = DbtPy.fetch_both(statment)
    while dataRecord:
        print("Tabid   : {}" . format(dataRecord[0]))
        print("Tabname : {}" . format(dataRecord['tabname']))
        # print(dataRecord)
        dataRecord = DbtPy.fetch_both(statment)
```

DbtPy.fetch_row

bool DbtPy.fetch_row (IFXStatement stmt [, int row_number])

描述

- 将结果集指针设置为下一行或请求的行。
- 使用DbtPy.fetch_row()用于遍历结果集，或者在请求可滚动游标时指向结果集中的特定行。
- 要从结果集中检索单个字段，请调用DbtPy.result()函数。而不是调用DbtPy.fetch_row()和DbtPy.result()，大多数应用程序将调用DbtPy.fetch_assoc()，DbtPy.fetch_both() 或 DbtPy.fetch_tuple() 中的一个来推进结果集指针并返回完整的行。

参数

- stmt - 包含结果集的有效stmt资源。
- row_number - 从结果集中请求特定的索引为1开始的行。如果结果集中使用只向前游标，传递此参数将导致警告。

返回值

- 如果请求的行存在于结果集中，则返回True。

- 如果请求的行不存在于结果集中，则返回False。

示例

参考代码： test_017_fetch_row.py

```
    statment = DbtPy.exec_immediate(conn, "select tabid,tabname from systables
where tabid < 10")
    bool = DbtPy.fetch_row(statment)
    if bool:
        print("Tabid   : {}" . format(DbtPy.result(statment,'tabid')))
        print("Tabname : {}" . format(DbtPy.result(statment,1)))
```

DbtPy.field_display_size

int DbtPy.field_display_size (IFXStatement stmt, mixed column)

描述

- 返回显示结果集中列所需的最大字节数。

参数

- stmt - 包含结果集的有效stmt资源。
- column - 指定结果集中的列。可以是表示列的0索引位置的整数，也可以是包含列名称的字符串。

返回值

- 返回显示指定列所需的最大字节数的整数值;
- 如果列不存在，则返回False。

示例

参考代码： test_30_num_fields.py

```
    statment = DbtPy.exec_immediate(conn, "select tabid,tabname from systables
limit 2")
    result = DbtPy.num_fields(statment)
    print("结果集中字段数量 :  {}" . format(result))

    for i in range(0, result) :
        print("field_name      : {}" . format(DbtPy.field_name(statment, i)))
        print("field_num       : {}" . format(DbtPy.field_num(statment, i)))
        print("field_type      : {}" . format(DbtPy.field_type(statment, i)))
        print("field_width     : {}" . format(DbtPy.field_width(statment, i)))
        print("field_precision : {}" . format(DbtPy.field_precision(statment, i)))
        print("field_scale     : {}" . format(DbtPy.field_scale(statment, i)))
        print("field_display_size    : {}" .
format(DbtPy.field_display_size(statment,i)))
```

DbtPy.field_name

string DbtPy.field_name (IFXStatement stmt, mixed column)

描述

- 返回结果集中指定列的名称。

参数

- stmt - 包含结果集的有效stmt资源。
- column - 指定结果集中的列。可以是表示列的0索引位置的整数，也可以是包含列名称的字符串。

返回值

- 返回一个包含指定列名称的字符串；
- 如果列不存在则返回False。

示例
参考代码： test_30_num_fields.py

```
    statment = DbtPy.exec_immediate(conn, "select tabid,tabname from systables
limit 2")
    result = DbtPy.num_fields(statment)
    print("结果集中字段数量 :  {}" . format(result))

    for i in range(0, result) :
        print("field_name      : {}" . format(DbtPy.field_name(statment, i)))
        print("field_num       : {}" . format(DbtPy.field_num(statment, i)))
        print("field_type      : {}" . format(DbtPy.field_type(statment, i)))
        print("field_width     : {}" . format(DbtPy.field_width(statment, i)))
        print("field_precision : {}" . format(DbtPy.field_precision(statment, i)))
        print("field_scale     : {}" . format(DbtPy.field_scale(statment, i)))
        print("field_display_size    : {}" .
format(DbtPy.field_display_size(statment,i)))
```

DbtPy.field_num

int DbtPy.field_num (IFXStatement stmt, mixed column)

描述

- 返回指定列在结果集中的位置。

参数

- stmt - 包含结果集的有效stmt资源。
- column - 指定结果集中的列。可以是表示列的0索引位置的整数，也可以是包含列名称的字符串。

返回值

- 返回一个整数，其中包含指定列的0索引位置；
- 如果列不存在，则返回False。

示例

参考代码： test_30_num_fields.py

```
    statement = DbtPy.exec_immediate(conn, "select tabid,tabname from systables
limit 2")
    result = DbtPy.num_fields(statment)
    print("结果集中字段数量 ：  {}" . format(result))

    for i in range(0, result) :
        print("field_name      : {}" . format(DbtPy.field_name(statment, i)))
        print("field_num       : {}" . format(DbtPy.field_num(statment, i)))
        print("field_type      : {}" . format(DbtPy.field_type(statment, i)))
        print("field_width     : {}" . format(DbtPy.field_width(statment, i)))
        print("field_precision : {}" . format(DbtPy.field_precision(statment, i)))
        print("field_scale     : {}" . format(DbtPy.field_scale(statment, i)))
        print("field_display_size    : {}" .
format(DbtPy.field_display_size(statment,i)))
```

DbtPy.field_precision

int DbtPy.field_precision (IFXStatement stmt, mixed column)

描述

- 返回结果集中指定列的精度。

参数

- stmt - 包含结果集的有效stmt资源。
- column - 指定结果集中的列。可以是表示列的0索引位置的整数，也可以是包含列名称的字符串。

返回值

- 返回一个包含指定列精度的整数；
- 如果列不存在，则返回False。

示例

参考代码： test_30_num_fields.py

```
    statement = DbtPy.exec_immediate(conn, "select tabid,tabname from systables
limit 2")
    result = DbtPy.num_fields(statment)
    print("结果集中字段数量 ：  {}" . format(result))

    for i in range(0, result) :
        print("field_name      : {}" . format(DbtPy.field_name(statment, i)))
        print("field_num       : {}" . format(DbtPy.field_num(statment, i)))
        print("field_type      : {}" . format(DbtPy.field_type(statment, i)))
        print("field_width     : {}" . format(DbtPy.field_width(statment, i)))
        print("field_precision : {}" . format(DbtPy.field_precision(statment, i)))
```

```
        print("field_scale     : {}" . format(DbtPy.field_scale(statment, i)))
        print("field_display_size    : {}" .
format(DbtPy.field_display_size(statment,i)))
```

DbtPy.field_scale

int DbtPy.field_scale (IFXStatement stmt, mixed column)

描述

- 返回结果集中指定列的比例。

参数

- stmt - 包含结果集的有效stmt资源。
- column - 指定结果集中的列。可以是表示列的0索引位置的整数，也可以是包含列名称的字符串。

返回值

- 返回一个包含指定列的比例的整数；
- 如果列不存在则返回False。

示例
参考代码： test_30_num_fields.py

```
    statment = DbtPy.exec_immediate(conn, "select tabid,tabname from systables
limit 2")
    result = DbtPy.num_fields(statment)
    print("结果集中字段数量 ：  {}" . format(result))

    for i in range(0, result) :
        print("field_name      : {}" . format(DbtPy.field_name(statment, i)))
        print("field_num       : {}" . format(DbtPy.field_num(statment, i)))
        print("field_type      : {}" . format(DbtPy.field_type(statment, i)))
        print("field_width     : {}" . format(DbtPy.field_width(statment, i)))
        print("field_precision : {}" . format(DbtPy.field_precision(statment, i)))
        print("field_scale     : {}" . format(DbtPy.field_scale(statment, i)))
        print("field_display_size    : {}" .
format(DbtPy.field_display_size(statment,i)))
```

DbtPy.field_type

string DbtPy.field_type (IFXStatement stmt, mixed column)

描述

- 返回结果集中指定列的数据类型。

参数

- stmt - 包含结果集的有效stmt资源。

- column - 指定结果集中的列。可以是表示列的0索引位置的整数，也可以是包含列名称的字符串。

返回值

- 返回一个字符串，其中包含指定列的定义数据类型；
- 如果列不存在，则返回False。

示例

参考代码： test_30_num_fields.py

```
    statment = DbtPy.exec_immediate(conn, "select tabid,tabname from systables
limit 2")
    result = DbtPy.num_fields(statment)
    print("结果集中字段数量 ： {}" . format(result))

    for i in range(0, result) :
        print("field_name      : {}" . format(DbtPy.field_name(statment, i)))
        print("field_num       : {}" . format(DbtPy.field_num(statment, i)))
        print("field_type      : {}" . format(DbtPy.field_type(statment, i)))
        print("field_width     : {}" . format(DbtPy.field_width(statment, i)))
        print("field_precision : {}" . format(DbtPy.field_precision(statment, i)))
        print("field_scale     : {}" . format(DbtPy.field_scale(statment, i)))
        print("field_display_size    : {}" .
format(DbtPy.field_display_size(statment,i)))
```

DbtPy.field_width

int DbtPy.field_width (IFXStatement stmt, mixed column)

描述

- 返回结果集中指定列的当前值的宽度。对于定长数据类型，这是列的最大宽度;对于变长数据类型，这是列的实际宽度。

参数

- stmt - 包含结果集的有效stmt资源。
- column - 指定结果集中的列。可以是表示列的0索引位置的整数，也可以是包含列名称的字符串。

返回值

- 返回一个包含指定字符或二进制列宽度的整数；
- 如果列不存在，则为False。

示例

参考代码： test_30_num_fields.py

```
    statment = DbtPy.exec_immediate(conn, "select tabid,tabname from systables
limit 2")
    result = DbtPy.num_fields(statment)
```

```
    print("结果集中字段数量 :  {}" . format(result))

    for i in range(0, result) :
        print("field_name      : {}" . format(DbtPy.field_name(statment, i)))
        print("field_num       : {}" . format(DbtPy.field_num(statment, i)))
        print("field_type      : {}" . format(DbtPy.field_type(statment, i)))
        print("field_width     : {}" . format(DbtPy.field_width(statment, i)))
        print("field_precision : {}" . format(DbtPy.field_precision(statment, i)))
        print("field_scale     : {}" . format(DbtPy.field_scale(statment, i)))
        print("field_display_size    : {}" .
format(DbtPy.field_display_size(statment,i)))
```

DbtPy.foreign_keys

IFXStatement DbtPy.foreign_keys (IFXConnection connection, string pk_qualifier, string pk_schema, string pk_table-name, string fk_qualifier, string fk_schema, string fk_table-name)

描述

- 返回列出表的外键的结果集。

参数

- connection - 有效的IFXConnection
- schema - 包含表的模式。如果schema为None，则使用连接的当前模式。
- table-name - 表名

返回值

- 返回一个IFXStatement，其结果集包含以下列：
  - PKTABLE_CAT - 包含主键的表的catalog名称。如果该表没有catalog，则该值为None。
  - PKTABLE_SCHEM - 包含主键的表的模式名。
  - PKTABLE_NAME - 包含主键的表的名称。
  - PKCOLUMN_NAME - 包含主键的列的名称。
  - FKTABLE_CAT - 包含外键的表的catalog名称。
  - FKTABLE_SCHEM - 包含外键的表的模式名。
  - FKTABLE_NAME - 包含外键的表的名称。
  - FKCOLUMN_NAME - 包含外键的列的名称
  - KEY_SEQ - 列在键中的1开始的索引位置。
  - UPDATE_RULE - 整数值，表示更新SQL操作时应用于外键的操作。
  - DELETE_RULE - 整数值，表示删除SQL操作时应用于外键的操作。
  - FK_NAME - 外键名称。
  - PK_NAME - 主键名称。

示例

参考代码：test_021_foreign_keys.py

```
    statment = DbtPy.foreign_keys(conn, None, None, None, None, None,
"tab_with_foreignkey")
```

```
result = DbtPy.fetch_tuple(statment)
# print(result)

print("PKTABLE_CAT    : {}" . format(result[0]))
print("PKTABLE_SCHEM  : {}" . format(result[1]))
print("PKTABLE_NAME   : {}" . format(result[2]))
print("PKCOLUMN_NAME  : {}" . format(result[3]))
print("FKTABLE_CAT    : {}" . format(result[4]))
print("FKTABLE_SCHEM  : {}" . format(result[5]))
print("FKTABLE_NAME   : {}" . format(result[6]))
print("FKCOLUMN_NAME  : {}" . format(result[7]))
print("KEY_SEQ        : {}" . format(result[8]))
print("UPDATE_RULE    : {}" . format(result[9]))
print("DELETE_RULE    : {}" . format(result[10]))
print("FK_NAME        : {}" . format(result[11]))
print("PK_NAME        : {}" . format(result[12]))
```

DbtPy.free_result

bool DbtPy.free_result (IFXStatement stmt)

描述

- 释放与结果集关联的系统和IFXConnections资源。这些资源在脚本结束时被隐式释放，但是您可以在脚本结束前调用DbtPy.free_result()来显式释放结果集资源。

参数

- stmt - 包含结果集的有效stmt资源。

返回值

- 成功返回True，失败返回False

示例
参考代码：test_019_free_result.py

```
bool = DbtPy.free_result(statment)
if bool :
    print("释放成功")
else :
    print("释放失败")
```

DbtPy.free_stmt

bool DbtPy.free_stmt (IFXStatement stmt) (DEPRECATED)

描述

- 释放与结果集关联的系统和IFXStatement资源。这些资源在脚本结束时被隐式释放，但是您可以在脚本结束前调用DbtPy.free_stmt()来显式释放结果集资源。

- 该API已弃用。应用程序应该使用DbtPy.free_result代替。

参数

- stmt - 包含结果集的有效stmt资源。

返回值

- 成功返回True，失败返回False

DbtPy.get_option

mixed DbtPy.get_option (mixed resc, int options, int type)

描述

- 返回连接或语句属性的当前设置的值。

参数

- resc - 有效的IFXConnection 或者 IFXStatement
- options - 要检索的选项
- type - 资源类型
    - 0 - IFXStatement
    - 1 - IFXConnection

返回值

- 返回所提供的资源属性的当前设置。

示例
参考代码： test_022_set_option.py

```
    print("设置conenct的可选参数")
    bool = DbtPy.set_option(conn, {DbtPy.ATTR_CASE: DbtPy.CASE_UPPER}, 1)
    if bool :
        print("设置connect的可选参数成功")
        result = DbtPy.get_option(conn, DbtPy.ATTR_CASE, 1)
        print("ATTR_CASE : {}" . format(str(result)))

    print("设置statment的可选参数")
    statment = DbtPy.prepare(conn, "select * from systables")
    bool = DbtPy.set_option(statment,  {DbtPy.SQL_ATTR_QUERY_TIMEOUT: 10}, 0)
    if bool :
        print("设置tatment的可选参数成功")
        result = DbtPy.get_option(statment, DbtPy.SQL_ATTR_CURSOR_TYPE, 0)
        print("SQL_ATTR_CURSOR_TYPE : {}" . format(str(result)))
```

DbtPy.num_fields

int DbtPy.num_fields (IFXStatement stmt)

描述

- 返回结果集中包含的字段的数量。这对于处理动态生成的查询返回的结果集或存储过程返回的结果集最有用，否则应用程序无法知道如何检索和使用结果。

参数

- stmt - 包含结果集的有效stmt资源。

返回值

- 返回一个整数值，表示与指定的IFXStatement相关联的结果集中字段的数量。
- 如果stmt不是一个有效的IFXStatement对象，则返回False。

示例

参考代码： test_30_num_fields.py

```
    statment = DbtPy.exec_immediate(conn, "select tabid,tabname from systables
limit 2")
    result = DbtPy.num_fields(statment)
    print("结果集中字段数量 ：  {}" . format(result))

    for i in range(0, result) :
        print("field_name      : {}" . format(DbtPy.field_name(statment, i)))
        print("field_num       : {}" . format(DbtPy.field_num(statment, i)))
        print("field_type      : {}" . format(DbtPy.field_type(statment, i)))
        print("field_width     : {}" . format(DbtPy.field_width(statment, i)))
        print("field_precision : {}" . format(DbtPy.field_precision(statment, i)))
        print("field_scale     : {}" . format(DbtPy.field_scale(statment, i)))
        print("field_display_size    : {}" .
format(DbtPy.field_display_size(statment,i)))
```

DbtPy.num_rows

int DbtPy.num_rows (IFXStatement stmt)

描述

- 返回SQL语句delete，insert或者update的行数。
- 要确定SELECT语句将返回的行数，请使用与预期的SELECT语句相同的谓词发出SELECT COUNT(*)并检索值。

参数

- stmt - 包含结果集的有效stmt资源。

返回值

- 返回受指定语句句柄发出的最后一条SQL语句影响的行数。

示例

参考代码： test_029_num_rows.py

```
    statement = DbtPy.exec_immediate(conn, "insert into tabmany select level,
'test'||level from dual connect by level < 1000")
    result = DbtPy.num_rows(statment)
    print("插入语句影响行数： {}" . format(result))

    prepare = DbtPy.prepare(conn, "delete from tabmany where col1 < ?")
    DbtPy.bind_param(prepare, 1, 100)
    result = DbtPy.execute(prepare)
    result = DbtPy.num_rows(prepare)
    print("删除语句影响行数： {}" . format(result))

    prepare = DbtPy.prepare(conn, "update tabmany set col2 = 'ttttt' where col1 <
?")
    DbtPy.bind_param(prepare, 1, 200)
    result = DbtPy.execute(prepare)
    result = DbtPy.num_rows(prepare)
    print("更新语句影响行数： {}" . format(result))
```

DbtPy.prepare

IFXStatement DbtPy.prepare (IFXConnection connection, string statement [, dict options])

描述

- 创建一个预编译的SQL语句，该语句可以包括0个或多个参数标记(?字符)表示输入、输出或输入/输出的参数。您可以使用DbtPy.bind_param()将参数传递给预编译的语句。或仅用于输入值，作为传递给DbtPy.execute()的元组。
- 在应用程序中使用准备好的语句有两个主要优点
  - 性能：预编译一条语句时，数据库服务器会创建一个优化的访问计划，以便使用该语句检索数据。随后使用DbtPy.execute()发出预编译的语句，使语句能够重用该访问计划，并避免为发出的每个语句动态创建新的访问计划的开销。
  - 安全：在预编译语句中，可以为输入值包括参数标记。当使用占位符的输入值执行准备好的语句时，数据库服务器会检查每个输入值，以确保类型与列定义或参数定义匹配。

参数

- connection - 有效的IFXConnection
- statement - SQL语句，可选地包含一个或多个参数标记。
- options - 包含语句选项的dict。
  - SQL_ATTR_CURSOR_TYPE - 将游标类型设置为以下类型之一(并非所有数据库都支持)
    - SQL_CURSOR_FORWARD_ONLY
    - SQL_CURSOR_KEYSET_DRIVEN
    - SQL_CURSOR_DYNAMIC
    - SQL_CURSOR_STATIC

返回值

- 如果数据库服务器成功地解析和准备了SQL语句，则返回一个IFXStatement对象；
- 如果数据库服务器返回错误，则返回False。

示例

参考代码： test_026_prepare.py

```
    statement = DbtPy.prepare(conn, "select tabid, tabname from systables where
tabid = ? and tabname = ?")
    DbtPy.bind_param(statment, 1, 1, DbtPy.SQL_PARAM_INPUT, DbtPy.SQL_INTEGER)
    DbtPy.bind_param(statment, 2, 'systables', DbtPy.SQL_PARAM_INPUT,
DbtPy.SQL_VARCHAR)
    DbtPy.execute(statment)
    result = DbtPy.fetch_tuple(statment)
    # print(result)
    print("Tabid   : {}" . format(str(result[0])))
    print("Tabname : {}" . format(str(result[1])))
```

DbtPy.primary_keys

IFXStatement DbtPy.primary_keys (IFXConnection connection, string qualifier, string schema, string table-name)

描述

- 返回列出表的主键的结果集。

参数

- connection - 有效的IFXConnection
- schema - 包含表的schema。如果schema为None，则使用连接的当前模式。
- table-name - 表名

返回值

- 返回一个IFXStatement，其结果集包含以下列：
  - TABLE_CAT - 包含主键的表的catalog名称。如果该表没有catalog，则该值为None。
  - TABLE_SCHEM - 包含主键的schema的名称。
  - TABLE_NAME - 包含主键的表的名称。
  - COLUMN_NAME - 包含主键的列的名称。
  - KEY_SEQ - 列在键中的从1开始索引的位置。
  - PK_NAME - 主键的名称

示例

参考代码： test_020_primary_keys.py

```
    statement = DbtPy.primary_keys(conn, None, None, "tab_with_primarykey")
    result = DbtPy.fetch_tuple(statment)
    # print(result)
    print("TABLE_CAT    : {}" . format(result[0]))
    print("TABLE_SCHEM  : {}" . format(result[1]))
    print("TABLE_NAME   : {}" . format(result[2]))
    print("COLUMN_NAME  : {}" . format(result[3]))
```

```
    print("KEY_SEQ      : {}" . format(result[4]))
    print("PK_NAME      : {}" . format(result[5]))
```

DbtPy.procedure_columns

IFXStatement DbtPy.procedure_columns (IFXConnection connection, string qualifier, string schema, string procedure, string parameter)

描述

- 返回一个结果集，列出一个或多个存储过程的参数

参数

- connection - 有效的IFXConnection
- schema - 包含过程的模式。该参数接受包含 _ 和 % 作为通配符的搜索模式。
- procedure - 存储过程的名称。该参数接受包含 _ 和 % 作为通配符的搜索模式。
- parameter - 参数名称。该参数接受包含 _ 和 % 作为通配符的搜索模式。如果该参数为None，返回所有的参数。

返回值

- 返回一个IFXStatement，其结果集包含以下列：
    - PROCEDURE_CAT - 包含存储过程的catalog名称。如果该存储过程没有catalog，则该值为None。
    - PROCEDURE_SCHEM - 包含存储过程的schema名称
    - PROCEDURE_NAME - 存储过程的名称。
    - COLUMN_NAME - 参数的名称。
    - COLUMN_TYPE - 表示参数类型的整数值：
        - 1 (SQL_PARAM_INPUT) - 输入参数 (IN).
        - 2 (SQL_PARAM_INPUT _OUTPUT) - 输入输出参数 (INOUT).
        - 3 (SQL_PARAM_OUTPUT) - 输出参数 (OUT).
    - DATA_TYPE - 表示为整数值的参数的SQL数据类型。
    - TYPE_NAME - 表示参数的数据类型的字符串。
    - COLUMN_SIZE - 表示参数大小的整数值。
    - BUFFER_LENGTH - 存储此参数的数据所需的最大字节数。
    - DECIMAL_DIGITS - 参数的刻度，如果刻度不适用，则为None。
    - NUM_PREC_RADIX - 一个整数值，可以是10(表示精确的数字数据类型)，2(表示近似的数字数据类型)，或者None(表示基数不适用的数据类型)。
    - NULLABLE - 一个整数值，表示参数是否可为空。

- REMARKS - 参数的描述。
- COLUMN_DEF - 参数的默认值。
- SQL_DATA_TYPE - 表示参数大小的整数值。
- SQL_DATETIME_SUB - 返回表示datetime子类型代码的整数值，对于不适用此方法的SQL数据类型，则返回None。
- CHAR_OCTET_LENGTH - 字符数据类型参数的最大字节长度，对于单字节字符集数据，该参数匹配COLUMN_SIZE，对于非字符数据类型，该参数为None。
- ORDINAL_POSITION - 参数在CALL语句中的以1开始为索引的位置。
- IS_NULLABLE - 一个字符串值，其中'YES'表示参数接受或返回无值，'NO'表示参数不接受或返回无值。

示例

参考代码：test_024_procedure_columns.py

```
statment = DbtPy.procedure_columns(conn, None, None, "proc1", None)
result = DbtPy.fetch_tuple(statment)
# print(result)
while result :
    print("PROCEDURE_CAT    : {}" . format(str(result[0])))
    print("PROCEDURE_SCHEM  : {}" . format(str(result[1])))
    print("PROCEDURE_NAME   : {}" . format(str(result[2])))
    print("COLUMN_NAME      : {}" . format(str(result[3])))
    print("COLUMN_TYPE      : {}" . format(str(result[4])))
    print("DATA_TYPE        : {}" . format(str(result[5])))
    print("TYPE_NAME        : {}" . format(str(result[6])))
    print("COLUMN_SIZE      : {}" . format(str(result[7])))
    print("BUFFER_LENGTH    : {}" . format(str(result[8])))
    print("DECIMAL_DIGITS   : {}" . format(str(result[9])))
    print("NUM_PREC_RADIX   : {}" . format(str(result[10])))
    print("NULLABLE         : {}" . format(str(result[11])))
    print("REMARKS          : {}" . format(str(result[12])))
    print("COLUMN_DEF       : {}" . format(str(result[13])))
    print("SQL_DATA_TYPE    : {}" . format(str(result[14])))
    print("SQL_DATETIME_SUB : {}" . format(str(result[15])))
    print("CHAR_OCTET_LENGTH: {}" . format(str(result[16])))
    print("ORDINAL_POSITION : {}" . format(str(result[17])))
    print("IS_NULLABLE      : {}" . format(str(result[18])))

    result = DbtPy.fetch_tuple(statment)
```

DbtPy.procedures

resource DbtPy.procedures (IFXConnection connection, string qualifier, string schema, string procedure)

描述

- 返回一个结果集，列出在数据库中注册的存储过程。

参数

- connection - 有效的IFXConnection
- schema - 包含过程的模式。该参数接受包含 _ 和 % 作为通配符的搜索模式。
- procedure - 存储过程的名称。该参数接受包含 _ 和 % 作为通配符的搜索模式。

返回值

- 返回一个IFXStatement，其结果集包含以下列：
  - PROCEDURE_CAT - 包含存储过程的catalog名称。如果该存储过程没有catalog，则该值为None。
  - PROCEDURE_SCHEM - 包含存储过程的schema名称
  - PROCEDURE_NAME - 存储过程的名称。
  - NUM_INPUT_PARAMS - 存储过程的输入参数 (IN) 的数目。
  - NUM_OUTPUT_PARAMS - 存储过程的输出参数 (OUT) 的数目。
  - NUM_RESULT_SETS - 存储过程返回的结果集的数目。
  - REMARKS - 存储过程的描述。
  - PROCEDURE_TYPE - 总是返回1，表示存储过程不返回返回值。

示例

参考代码： test_023_procedures.py

```
    statment = DbtPy.procedures(conn, None, None, "proc1")
    result = DbtPy.fetch_tuple(statment)
    # print(result)
    print("PROCEDURE_CAT    : {}" . format(str(result[0])))
    print("PROCEDURE_SCHEM  : {}" . format(str(result[1])))
    print("PROCEDURE_NAME   : {}" . format(str(result[2])))
    print("NUM_INPUT_PARAMS : {}" . format(str(result[3])))
    print("NUM_OUTPUT_PARAMS: {}" . format(str(result[4])))
    print("NUM_RESULT_SETS  : {}" . format(str(result[5])))
    print("REMARKS          : {}" . format(str(result[6])))
    print("PROCEDURE_TYPE   : {}" . format(str(result[7])))
```

DbtPy.result

mixed DbtPy.result (IFXStatement stmt, mixed column)

描述

- 使用DbtPy.result()返回结果集中当前row的指定列的值。你必须调用DbtPy。在调用DbtPy.result()之前调用DbtPy.fetch_row()来设置结果集指针的位置。

参数

- stmt - 包含结果集的有效stmt资源。
- column - 映射到结果集中以0开始的索引的字段的整数，或者匹配列名称的字符串。

返回值

- 如果结果集中存在请求的字段，则返回该字段的值。
- 如果该字段不存在，则返回None，并发出警告。

示例
参考代码： test_018_result.py

```
    statment = DbtPy.exec_immediate(conn, "select tabid,tabname from systables
where tabid < 10")
    bool = DbtPy.fetch_row(statment)
    if bool:
        print("Tabid   : {}" . format(DbtPy.result(statment,'tabid')))
        print("Tabname : {}" . format(DbtPy.result(statment,1)))
```

DbtPy.rollback

bool DbtPy.rollback (IFXConnection connection)

描述

- 回滚指定的IFXConnection上正在进行的事务，并开始一个新的事务。
- Python应用程序通常默认为自动提交模式，因此DbtPy.rollback()通常没有效果，除非在IFXConnection中关闭了自动提交。
- 注意:如果指定的IFXConnection是一个持久连接，那么使用该持久连接的所有应用程序的所有正在进行的事务都将回滚。因此，不建议在需要事务的应用程序中使用持久连接。

参数

- connection - 有效的IFXConnection

返回值

- 成功返回True，不成功返回False。

示例
参考代码：test_008_rollback.py

```
commit = DbtPy.autocommit(conn, DbtPy.SQL_AUTOCOMMIT_OFF)
    if commit :
        print("设置自动提交状态为OFF成功")
    else :
        print("设置自动提交状态为OFF失败")

    statment = DbtPy.exec_immediate(conn, "insert into tab1(col1) values(1)")
    rollback = DbtPy.rollback(conn)
    if rollback :
        print("回滚成功")
    else :
        print("回滚失败")
```

DbtPy.server_info

IFXServerInfo DbtPy.server_info (IFXConnection connection)

描述

- 返回一个只读对象，其中包含有关GBase 8s服务器的信息。

参数

- connection - 有效的IFXConnection

返回值

- 成功时，一个包含以下字段的对象：
    - DBMS_NAME - 连接到的数据库服务器的名称。
    - DBMS_VER - 数据库的版本号，格式为"MM.mm.uuuu"，其中 MM 是主版本号，mm 是次版本号，uuuu 是更新版本号。例："08.02.0001"
    - DB_NAME - 连接到的数据库的名称。(string)
    - DFT_ISOLATION - 服务器支持的默认事务隔离级别: (string)
        - UR - Uncommitted read: 所有并发事务都可以立即看到更改。
        - CS - Cursor stability: 一个事务读取的行可以被第二个并发事务修改和提交。
        - RS - Read stability: 事务可以添加或删除匹配搜索条件或待处理事务的行。
        - RR - Repeatable read: 受待处理事务影响的数据对其他事务不可用。
        - NC - No commit: 在成功的操作结束时，任何更改都是可见的。不允许显式提交和回滚。
    - IDENTIFIER_QUOTE_CHAR - 用于分隔标识符的字符。 (string)
    - INST_NAME - 包含数据库的数据库服务器上的实例名称。 (string)
    - ISOLATION_OPTION - 数据库服务器支持的隔离级别元组。隔离级别在DFT_ISOLATION属性中进行了描述。 (tuple)
    - KEYWORDS - 数据库服务器保留的关键字的元组。(tuple)
    - LIKE_ESCAPE_CLAUSE - 如果数据库服务器支持使用%和_通配符，则为True。如果数据库服务器不支持这些通配符，则为False。(bool)
    - MAX_COL_NAME_LEN - 数据库服务器支持的列名的最大长度，单位为字节。(int)
    - MAX_IDENTIFIER_LEN - 数据库服务器支持的SQL标识符的最大长度，以字符表示。(int)
    - MAX_INDEX_SIZE - 数据库服务器支持的索引中合并列的最大大小(以字节表示)。(int)
    - MAX_PROC_NAME_LEN - 数据库服务器支持的过程名的最大长度，以字节表示。(int)
    - MAX_ROW_SIZE - 数据库服务器支持的基表中一行的最大长度，以字节表示。(int)
    - MAX_SCHEMA_NAME_LEN - 数据库服务器支持的模式名的最大长度，以字节表示。(int)

    - MAX_STATEMENT_LEN - 数据库服务器支持的SQL语句的最大长度，以字节表示。(int)
    - MAX_TABLE_NAME_LEN - 数据库服务器支持的表名的最大长度，以字节表示。(int)
    - NON_NULLABLE_COLUMNS - 如果数据库服务器支持定义为NOT NULL的列，则为True;如果数据库服务器不支持定义为NOT NULL的列，则为False。(bool)
    - PROCEDURES - 如果数据库服务器支持使用CALL语句调用存储过程，则为True;如果数据库服务器不支持CALL语句，则为False。(bool)
    - SPECIAL_CHARS - 包含除A- z、0-9和下划线之外的所有可用于标识符名称的字符串。(string)
    - SQL_CONFORMANCE - 数据库服务器提供的符合ANSI或ISO SQL-92规范的级别:(string)
        - ENTRY - 入门级SQL-92兼容性。
        - FIPS127 - FIPS-127-2过渡兼容性。
        - FULL - 完全SQL-92兼容。
        - INTERMEDIATE - 中性SQL-92兼容

- 失败时，返回False

示例

参考代码：test_005_server_info.py

```
server = DbtPy.server_info(conn)
if server:
  print("服务端信息: ")
  print("DBMS_NAME: \"%s\"" % server.DBMS_NAME)
  print("DBMS_VER: \"%s\"" % server.DBMS_VER)
  print("DB_NAME: \"%s\"" % server.DB_NAME)
  print("INST_NAME: \"%s\"" % server.INST_NAME)
  print("SPECIAL_CHARS: \"%s\"" % server.SPECIAL_CHARS)
  print("KEYWORDS: int(%d)" % len(server.KEYWORDS))
  print("DFT_ISOLATION: \"%s\"" % server.DFT_ISOLATION)
  il = ''
  for opt in server.ISOLATION_OPTION:
    il += opt + " "
  print("ISOLATION_OPTION: \"%s\"" % il)
  print("SQL_CONFORMANCE: \"%s\"" % server.SQL_CONFORMANCE)
  print("PROCEDURES:", server.PROCEDURES)
  print("IDENTIFIER_QUOTE_CHAR: \"%s\"" % server.IDENTIFIER_QUOTE_CHAR)
  print("LIKE_ESCAPE_CLAUSE:", server.LIKE_ESCAPE_CLAUSE)
  print("MAX_COL_NAME_LEN: int(%d)" % server.MAX_COL_NAME_LEN)
  print("MAX_ROW_SIZE: int(%d)" % server.MAX_ROW_SIZE)
  print("MAX_IDENTIFIER_LEN: int(%d)" % server.MAX_IDENTIFIER_LEN)
  print("MAX_INDEX_SIZE: int(%d)" % server.MAX_INDEX_SIZE)
  print("MAX_PROC_NAME_LEN: int(%d)" % server.MAX_PROC_NAME_LEN)
  print("MAX_SCHEMA_NAME_LEN: int(%d)" % server.MAX_SCHEMA_NAME_LEN)
  print("MAX_STATEMENT_LEN: int(%d)" % server.MAX_STATEMENT_LEN)
  print("MAX_TABLE_NAME_LEN: int(%d)" % server.MAX_TABLE_NAME_LEN)
  print("NON_NULLABLE_COLUMNS:", server.NON_NULLABLE_COLUMNS)
```

```
        DbtPy.close(conn)
    else:
        print("服务端信息信息错误.")
```

DbtPy.set_option

bool DbtPy.set_option (mixed resc, dict options, int type)

描述

- 为IFXConnection 或者 IFXStatement设置选项。不能为结果集资源设置选项。

参数

- resc - 有效的IFXConnection 或者 IFXStatement.
- options - 要设置的选项
- type - 指定resc类型的字段
    - 0 - IFXStatement
    - 1 - IFXConnection

返回值

- 成功返回True，不成功返回False。

示例
参考代码： test_022_set_option.py

```
print("设置conenct的可选参数")
bool = DbtPy.set_option(conn, {DbtPy.ATTR_CASE: DbtPy.CASE_UPPER}, 1)
if bool :
    print("设置connect的可选参数成功")
    result = DbtPy.get_option(conn, DbtPy.ATTR_CASE, 1)
    print("ATTR_CASE : {}" . format(str(result)))

print("设置statment的可选参数")
statment = DbtPy.prepare(conn, "select * from systables")
bool = DbtPy.set_option(statment,  {DbtPy.SQL_ATTR_QUERY_TIMEOUT: 10}, 0)
if bool :
    print("设置tatment的可选参数成功")
    result = DbtPy.get_option(statment, DbtPy.SQL_ATTR_CURSOR_TYPE, 0)
    print("SQL_ATTR_CURSOR_TYPE : {}" . format(str(result)))
```

DbtPy.special_columns

IFXStatement DbtPy.special_columns (IFXConnection connection, string qualifier, string schema, string table_name, int scope)

描述

- 返回一个结果集，列出表的唯一行标识符列。

参数

- connection - 有效的IFXConnection
- schema - 表所有的schema
- table_name - 表名
- scope - 表示唯一行标识符有效的最小持续时间的整数值。这可以是以下值之一：
    - 0 - 行标识符仅在游标位于行上时有效。(SQL_SCOPE_CURROW)
    - 1 - 行标识符在事务的持续时间内有效。(SQL_SCOPE_TRANSACTION)
    - 2 - 行标识符在连接期间有效。(SQL_SCOPE_SESSION)

返回值

- 返回一个IFXStatement，其结果集包含以下列：
    - SCOPE - 表示唯一行标识符有效的最小持续时间的整数值
        - 0 - 行标识符仅在游标位于行上时有效。(SQL_SCOPE_CURROW)
        - 1 - 行标识符在事务的持续时间内有效。(SQL_SCOPE_TRANSACTION)
        - 2 - 行标识符在连接期间有效。(SQL_SCOPE_SESSION)
    - COLUMN_NAME - 唯一列的名称。
    - DATA_TYPE - 列的SQL数据类型。
    - TYPE_NAME - 列的SQL数据类型的字符串表示形式。
    - COLUMN_SIZE - 表示列大小的整数值。
    - BUFFER_LENGTH - 存储这个列存储数据所需的最大字节数。
    - DECIMAL_DIGITS - 列的刻度，如果不适用刻度，则为None。
    - PSEUDO_COLUMN - 总是返回 1。

示例

参考代码： test_031_special_columns.py

```
    statment = DbtPy.exec_immediate(conn, "create table tabspecial(col1 int, col2
varchar(20), primary key(col2))")
    statment = DbtPy.special_columns(conn, None, None, "tabspecial", 0)
    result = DbtPy.fetch_assoc(statment)
    while result :
        # print(result)
        print("COLUMN_NAME     : {}" . format(result['COLUMN_NAME']))
        print("DATA_TYPE       : {}" . format(result['DATA_TYPE']))
        print("TYPE_NAME       : {}" . format(result['TYPE_NAME']))
        print("COLUMN_SIZE     : {}" . format(result['COLUMN_SIZE']))
        print("BUFFER_LENGTH   : {}" . format(result['BUFFER_LENGTH']))
        print("DECIMAL_DIGITS  : {}" . format(result['DECIMAL_DIGITS']))
        print("SCOPE           : {}" . format(result['SCOPE']))
        print("PSEUDO_COLUMN   : {}" . format(result['PSEUDO_COLUMN']))
        result = DbtPy.fetch_assoc(statment)
```

DbtPy.statistics

IFXStatement DbtPy.statistics (IFXConnection connection, string qualifier, string schema, string table_name, bool unique)

描述

- 返回一个结果集，列出表的索引和统计信息。

参数

- connection - 有效的IFXConnection
- schema - 包含表的schema。如果该参数为None，则返回当前用户模式的统计信息和索引。
- table_name - 表名。
- unique - 一个布尔值，表示要返回的索引信息的类型。
    - False - 只返回表上惟一索引的信息。
    - True - 返回表中所有索引的信息。

返回值

- 返回一个IFXStatement，其结果集包含以下列：
    - TABLE_CAT - 包含表格的catalog。如果该表没有catalog，则该值为None。
    - TABLE_SCHEM - 包含表的模式的名称。
    - TABLE_NAME - 表名。
    - NON_UNIQUE - 一个整数值，表示索引是否禁止唯一值，或者行是否表示表本身的统计信息：
        - 0 (SQL_FALSE) - 索引允许重复的值。
        - 1 (SQL_TRUE) - 索引值必须唯一。
        - None - 这一行是表本身的统计信息。
    - INDEX_QUALIFIER - 表示限定符的字符串值，该限定符必须预先添加到INDEX_NAME以完全限定索引。
    - INDEX_NAME - 表示索引名称的字符串。
    - TYPE - 一个整数值，表示结果集中这一行中包含的信息的类型：
        - 0 (SQL_TABLE_STAT) - 该行包含有关表本身的统计信息。
        - 1 (SQL_INDEX_CLUSTERED) - 该行包含关于聚集索引的信息。
        - 2 (SQL_INDEX_HASH) - 该行包含有关散列索引的信息。
        - 3 (SQL_INDEX_OTHER) - 该行包含有关既没有聚集也没有散列的索引类型的信息。
    - ORDINAL_POSITION - 索引中列的以1为开始的索引位置。如果行包含有关表本身的统计信息，则为None。
    - COLUMN_NAME - 索引中列的名称。如果行包含有关表本身的统计信息，则为None。

- ASC_OR_DESC - A表示列按升序排序，D表示列按降序排序，如果行包含关于表本身的统计信息，则为None。
- CARDINALITY - 如果行包含有关索引的信息，则此列包含一个整数值，表示索引中惟一值的数目。如果行包含关于表本身的信息，则此列包含一个整数值，表示表中的行数。
- PAGES - 如果行包含有关索引的信息，则此列包含一个整数值，表示用于存储索引的页数。如果行包含关于表本身的信息，则此列包含一个整数值，表示用于存储表的页数。
- FILTER_CONDITION - 总是返回None。

示例

参考代码： test_032_statistics.py

```
statment = DbtPy.statistics(conn, None, None, "tabstatistics", True)
result = DbtPy.fetch_assoc(statment)
while result :
    # print(result)
    print("TABLE_CAT        : {}" . format(result['TABLE_CAT']))
    print("TABLE_SCHEM      : {}" . format(result['TABLE_SCHEM']))
    print("TABLE_NAME       : {}" . format(result['TABLE_NAME']))
    print("NON_UNIQUE       : {}" . format(result['NON_UNIQUE']))
    print("INDEX_QUALIFIER  : {}" . format(result['INDEX_QUALIFIER']))
    print("INDEX_NAME       : {}" . format(result['INDEX_NAME']))
    print("TYPE             : {}" . format(result['TYPE']))
    print("ORDINAL_POSITION : {}" . format(result['ORDINAL_POSITION']))
    print("COLUMN_NAME      : {}" . format(result['COLUMN_NAME']))
    print("ASC_OR_DESC      : {}" . format(result['ASC_OR_DESC']))
    print("CARDINALITY      : {}" . format(result['CARDINALITY']))
    print("PAGES            : {}" . format(result['PAGES']))
    print("FILTER_CONDITION : {}" . format(result['FILTER_CONDITION']))
    print("")

    result = DbtPy.fetch_assoc(statment)
```

DbtPy.stmt_error

string DbtPy.stmt_error ([IFXStatement stmt])

描述

- 当没有传递任何参数时，返回表示上次通过IFXStatement执行DbtPy.prepare(), DbtPy.exec_immediate() 或者 DbtPy.callproc() 返回的SQLSTATE
- 当传递一个有效的IFXStatement时，返回SQLSTATE，表示上次使用资源的操作失败的原因。

参数

- stmt - 有效的IFXStatement.

返回值

- 返回包含SQLSTATE值的字符串，如果没有错误，则返回空字符串。

DbtPy.stmt_errormsg

string DbtPy.stmt_errormsg ([IFXStatement stmt])

描述

- 当没有传递任何参数时，返回表示上次通过IFXStatement执行DbtPy.prepare(), DbtPy.exec_immediate() 或者 DbtPy.callproc() 返回的SQLCODE及错误信息
- 当传递一个有效的IFXStatement时，返回SQLCODE及错误信息，表示上次使用资源的操作失败的原因。

参数

- stmt - 有效的IFXStatement.

返回值

- 返回包含SQLCODE值的字符串，如果没有错误，则返回空字符串。

DbtPy.table_privileges

IFXStatement DbtPy.table_privileges (IFXConnection connection [, string qualifier [, string schema [, string table_name]]])

描述

- 返回一个结果集，列出数据库中的表和相关权限。

参数

- connection - 有效的IFXConnection
- schema - 包含表的模式。该参数接受包含_和%作为通配符的搜索模式。
- table_name - 表名。该参数接受包含_和%作为通配符的搜索模式。

返回值

- 返回一个IFXStatement，其结果集包含以下列：
  - TABLE_CAT - 包含表的catalog。如果该表没有catalog，则该值为None。
  - TABLE_SCHEM - 包含表的schema。
  - TABLE_NAME - 表名。
  - GRANTOR - 授予权限者。
  - GRANTEE - 被授权者。
  - PRIVILEGE - 被授予的权限。这可以是ALTER、CONTROL、DELETE、INDEX、INSERT、REFERENCES、SELECT或UPDATE之一。
  - IS_GRANTABLE - 字符串值“YES”或“NO”，表示被授权人是否可以将该权限授予其他用户。

示例

参考代码：test_010_table_privilegs.py

```
    statment = DbtPy.table_privileges(conn, None, None, "tab1")
    dataRecord = DbtPy.fetch_assoc(statment)
```

```
    while dataRecord:
        print("TABLE_CAT   : {}" .format(dataRecord['TABLE_CAT']))
        print("TABLE_SCHEM : {}" .format(dataRecord['TABLE_SCHEM']))
        print("TABLE_NAME  : {}" .format(dataRecord['TABLE_NAME']))
        print("GRANTOR     : {}" .format(dataRecord['GRANTOR']))
        print("GRANTEE     : {}" .format(dataRecord['GRANTEE']))
        print("PRIVILEGE   : {}" .format(dataRecord['PRIVELEGE']))
        print("IS_GRANTABLE: {}" .format(dataRecord['IS_GRANTABLE']))
        # print(dataRecord)
        dataRecord = DbtPy.fetch_assoc(statment)
```

DbtPy.tables

IFXStatement DbtPy.tables (IFXConnection connection [, string qualifier [, string schema [, string table-name [, string table-type]]]])

描述

- 返回一个结果集，列出数据库中的表和相关元数据

参数

- connection - 有效的IFXConnection
- schema - 包含表的模式。该参数接受包含_和%作为通配符的搜索模式。
- table-name - 表名。该参数接受包含_和%作为通配符的搜索模式。
- table-type -以逗号分隔的表类型标识符列表。要匹配所有表类型，请传递None或空字符串。
    - ALIAS
    - HIERARCHY TABLE
    - INOPERATIVE VIEW
    - NICKNAME
    - MATERIALIZED QUERY TABLE
    - SYSTEM TABLE
    - TABLE
    - TYPED TABLE
    - TYPED VIEW
    - VIEW

返回值

返回一个IFXStatement，其结果集包含以下列：

- TABLE_CAT - 包含表的catalog。如果该表没有catalog，则该值为None。
- TABLE_SCHEMA - 包含表的模式的名称。
- TABLE_NAME - 表名。
- TABLE_TYPE - 表的表类型标识符。
- REMARKS - 表的描述。

示例

参考代码：test_009_tables.py

```
statment = DbtPy.tables(conn, None, None, "tab1", None)
dataRecord = DbtPy.fetch_assoc(statment)
while dataRecord:
    print("TABLE_CAT   : {}" .format(dataRecord['TABLE_CAT']))
    print("TABLE_SCHEM : {}" .format(dataRecord['TABLE_SCHEM']))
    print("TABLE_NAME  : {}" .format(dataRecord['TABLE_NAME']))
    print("TABLE_TYPE  : {}" .format(dataRecord['TABLE_TYPE']))
    print("REMARKS     : {}" .format(dataRecord['REMARKS']))
    dataRecord = DbtPy.fetch_assoc(statment)
```

DbtPy当前支持的可选连接和语句参数

SQL_ATTR_AUTOCOMMIT： 自动提交
ATTR_CASE： 参数大小写
SQL_ATTR_CURSOR_TYPE： 游标类型
SQL_ATTR_QUERY_TIMEOUT： 查询超时

DbtPy对数据类型的支持

常用的数据类型如下：

字符类型

内置字符类型包括：char, nchar, varchar, nvarchar 和lvarchar
CHARACTER(n) 和 CHARACTER VARYING(n)这样的别名同样支持
参考代码：test_string_type.py

```
create = """
create table tab_string(
    col1 char(32767),
    col2 nchar(32767),
    col3 varchar(255),
    col4 nvarchar(255),
    col5 lvarchar(32739)
)
"""
stmt = DbtPy.exec_immediate(conn, create)
insert = """
insert into tab_string(col1, col2, col3, col4, col5) values
('字符字段char类型','本地化字符字段nchar类型','可变长度字符字段varchar类型',
'本地化可变长度字符字段nvarchar类型','扩展可变长度字符字段lvarchar类型')
"""
stmt = DbtPy.exec_immediate(conn, insert)

select = "select * from tab_string"
stmt = DbtPy.exec_immediate(conn, select)
result = DbtPy.fetch_assoc(stmt)
while result :
    print("字段1的字节长度为：{} ,值为：\"{}\"".
```

```
format(len(result['col1'].encode('utf8')),result['col1']))
        print("字段2的字节长度为: {} ,值为: \"{}\"".
format(len(result['col2'].encode('utf8')),result['col2']))
        print("字段3的字节长度为: {} ,值为: \"{}\"".
format(len(result['col3'].encode('utf8')),result['col3']))
        print("字段4的字节长度为: {} ,值为: \"{}\"".
format(len(result['col4'].encode('utf8')),result['col4']))
        print("字段5的字节长度为: {} ,值为: \"{}\"".
format(len(result['col5'].encode('utf8')),result['col5']))
        result = DbtPy.fetch_assoc(stmt)
```

数值类型

内置的数值类型包括:
整型：smallint, integer, int8, bigint 自增长整型：serial, serial8, bigserail
浮点型：smallfloat, float, decimal(P)
精确值型：decimal(P,S)
货币类型：money(P,S)
DEC(p,s)、NUMERIC(p,s)、INT和DOUBLE PRECISION这样的别名同样支持
参考代码：test_numeric_type.py

```
    create = """
    create table tab_numeric(
        col1 serial not null,
        col2 smallint,
        col3 int,
        col4 int8,
        col5 bigint,
        col6 smallfloat,
        col7 float,
        col8 decimal(32,20),
        col9 decimal(32),
        colx money(32,20)
    )
    """
    stmt = DbtPy.exec_immediate(conn, create)
    insert = """
    insert into tab_numeric(col1, col2, col3, col4, col5, col6, col7, col8, col9,
colx) values

(0,32767,2147483647,9223372036854775807,9223372036854775807,1234567890,12345678901
234567890,123456789012.123456789,12345678901234567890123456789012,123456789012.123
456789)
    """
    stmt = DbtPy.exec_immediate(conn, insert)

    select = "select * from tab_numeric"
    stmt = DbtPy.exec_immediate(conn, select)
    result = DbtPy.fetch_assoc(stmt)
    while result :
```

```
        print("字段 1的类型serail         ,值为: {}" . format(result['col1']))
        print("字段 2的类型samllint       ,值为: {}" . format(result['col2']))
        print("字段 3的类型integer        ,值为: {}" . format(result['col3']))
        print("字段 4的类型int8           ,值为: {}" . format(result['col4']))
        print("字段 5的类型bigint         ,值为: {}" . format(result['col5']))
        print("字段 6的类型smallfloat     ,值为: {}" . format(result['col6']))
        print("字段 7的类型float          ,值为: {}" . format(result['col7']))
        print("字段 8的类型decimal(32,20) ,值为: {}" . format(result['col8']))
        print("字段 9的类型decimal(32)    ,值为: {}" . format(result['col9']))
        print("字段10的类型money(32,20)   ,值为: {}" . format(result['colx']))
        result = DbtPy.fetch_assoc(stmt)
```

日期时间型

日期型：date
日期时间型：datetime [first to last]
间隔类型：interval [first to list]
TIMESTAMP(n)这样的别名在部分版本中支持
interval类型部分支持（实现支持部分间隔类型）

```
    create = """
    create table tab_datetime(
        col1 serial not null,
        col2 date,
        col3 datetime year to day,
        col4 datetime year to second,
        col5 datetime year to fraction(5),
        col6 interval day to minute
    )
    """
    stmt = DbtPy.exec_immediate(conn, create)
    prepare = """
    insert into tab_datetime(col1,col2,col3,col4,col5,col6) values(0,?,?,?,?,?)
    """
    stmt = DbtPy.prepare(conn, prepare)
    DbtPy.bind_param(stmt, 1, '2023-03-21')
    DbtPy.bind_param(stmt, 2, '2023-03-21')
    DbtPy.bind_param(stmt, 3, '2023-03-21 12:34:56')
    DbtPy.bind_param(stmt, 4, '2023-03-21 12:34:56.98765')
    DbtPy.bind_param(stmt, 5, '-3 12:13')
    result = DbtPy.execute(stmt)

    # 不支持interval year to month, day to second等
    select = "select * from tab_datetime"
    stmt = DbtPy.exec_immediate(conn, select)
    result = DbtPy.fetch_assoc(stmt)
    while result :
        print("字段 1的类型serail                ,值为: {}" .
format(result['col1']))
        print("字段 2的类型date                  ,值为: {}" .
format(result['col2']))
```

```
        print("字段 3的类型datetime Y-m-d          ,值为: {}" .
format(result['col3']))
        print("字段 4的类型datetime Y-m-d H:M:S    ,值为: {}" .
format(result['col4']))
        print("字段 5的类型datetime Y-m-d H:M:S.F5 ,值为: {}" .
format(result['col5']))
        print("字段 6的类型interval d H:M          ,值为: {}" .
format(result['col6']))
        result = DbtPy.fetch_assoc(stmt)
```

大对象和智能大对象类型

大对象类型：byte、text
智能大对象类型：blob、 clob
限制：
text仅支持insert/update，不支持查询
clob仅支持使用函数操作(filetoclob(),locopy(),dbms_lob_new_clob())
byte支持使用bytes操作
blob支持使用bytes和函数操作(filetoblob(),locopy())
dbms_lob_new_clob函数:

```
create function if not exists dbms_lob_new_clob (lvarchar)
returns clob with (not variant)
external name '$GBASEDBTDIR/extend/excompat.1.0/excompat.bld(dbms_lob_new_clob)'
language c;
```

参考代码：test_lob_type.py

```
    create = """
    create table tab_lob(
        col1 serial not null,
        col2 byte,
        col3 blob,
        col4 clob,
        col5 text
    )
    """
    stmt = DbtPy.exec_immediate(conn, create)
    # 使用了dbms_log_new_clob函数将string转换为clob
    prepare = "insert into tab_lob(col1,col2,col3,col4,col5)
values(0,?,?,dbms_lob_new_clob(?),?)"
    col2_byte = bytes('byte类型输入', encoding='UTF-8')
    col3_blob = bytes('blob类型输入', encoding='UTF-8')
    col4_clob = 'clob类型输入'
    col5_text = 'text类型输入'
    stmt = DbtPy.prepare(conn, prepare)
    DbtPy.bind_param(stmt,1, col2_byte, DbtPy.SQL_PARAM_INPUT)
    DbtPy.bind_param(stmt,2, col3_blob, DbtPy.SQL_PARAM_INPUT)
```

```
DbtPy.bind_param(stmt,3, col4_clob, DbtPy.SQL_PARAM_INPUT)
DbtPy.bind_param(stmt,4, col5_text, DbtPy.SQL_PARAM_INPUT)
result = DbtPy.execute(stmt)

# 暂时还不支持获取操作
select = "select col1,col2,col3,col4 from tab_lob"
stmt = DbtPy.exec_immediate(conn, select)
result = DbtPy.fetch_assoc(stmt)
while result :
    print("字段1的值为: \"{}\"". format(result['col1']))
    print("字段2的值为: \"{}\"". format(result['col2'].decode('UTF-8')))
    print("字段3的值为: \"{}\"". format(result['col3'].decode('UTF-8')))
    print("字段4的值为: \"{}\"". format(result['col4']))

    result = DbtPy.fetch_assoc(stmt)
```

布尔类型

布尔类型：boolean
取值范围：'t'/1, 'f'/0, null(None)
参考代码：test_boolean_type.py

```
create = """
create table tab_boolean(
    col1 serial not null,
    col2 boolean
)
"""
stmt = DbtPy.exec_immediate(conn, create)
prepare = """
insert into tab_boolean(col1,col2) values(0,?)
"""
stmt = DbtPy.prepare(conn, prepare)
DbtPy.bind_param(stmt, 1, True)
result = DbtPy.execute(stmt)
DbtPy.bind_param(stmt, 1, False)
result = DbtPy.execute(stmt)
DbtPy.bind_param(stmt, 1, None)
result = DbtPy.execute(stmt)

select = "select * from tab_boolean"
stmt = DbtPy.exec_immediate(conn, select)
result = DbtPy.fetch_assoc(stmt)
while result :
    print("字段 1的类型serail   ,值为: {}" . format(result['col1']))
    print("字段 2的类型boolean  ,值为: {}" . format(result['col2']))
    result = DbtPy.fetch_assoc(stmt)
```

扩展类型LIST

LIST类型：list(TYPE not null)
扩展类型查询结果为bytes类型，需要decode操作
参考代码：test_list_type.py

```
    create = """
    create table tab_list(
        col1 serial not null,
        col2 LIST(varchar(20) not null)
    )
    """
    stmt = DbtPy.exec_immediate(conn, create)
    prepare = """
    insert into tab_list(col1,col2) values(0,?)
    """
    col2_list = None
    stmt = DbtPy.prepare(conn, prepare)
    DbtPy.bind_param(stmt, 1, col2_list, DbtPy.SQL_PARAM_INPUT, DbtPy.SQL_CHAR,
DbtPy.SQL_INFX_RC_COLLECTION)
    col2_list = "LIST{'aaaa','bbbb','cccc'}"
    result = DbtPy.execute(stmt,(col2_list,))

    select = "select * from tab_list"
    stmt = DbtPy.exec_immediate(conn, select)
    result = DbtPy.fetch_assoc(stmt)
    while result :
        print("字段 1的类型serail   ,值为：{}" . format(result['col1']))
        # LIST 类型以bytes输出，需转码成字符串
        print("字段 2的类型list     ,值为：{}" . format(result['col2'].decode('UTF-
8')))
        result = DbtPy.fetch_assoc(stmt)
```