PUBLIC

SDK for SAP Adaptive Server Enterprise 16.0 SP03 PL06

Document Version: 1.0 – 2019-1-15

Programmers Guide for Python

© 2018 SAP SE or an SAP affiliate company. All rights reserved.

THE BEST RUN SAP

Content

1 SAP Adaptive Server Enterprise Extension Module for Python

The extension module for Python, `sybpydb`, provides an SAP specific Python interface that is used to execute queries against an SAP Adaptive Server Enterprise (ASE) database.

The extension module implements the Python Database API specification version 2.0 with extensions. For more information about the API specification, see http://www.python.org/dev/peps/pep-0249 .

Related Information

Required Components [page 3]
Version Requirements [page 3]
Extension Module for Python Installation Types [page 4]
Extension Module for Python Configuration Overview [page 4]
Developing Python Applications [page 6]
Extension Module for Python API Reference [page 14]

1.1 Required Components

Access to an SAP Adaptive Server Enterprise database using the Python programming language requires these components.

- `sybpydb` – extension module for the Python scripting language.
- Open Client SDK – provides application development tools that allow access to data source, information application or system service.

1.2 Version Requirements

SAP Adaptive Server Enterprise Extension Module for Python has these version requirements.

- Adaptive Server Enterprise – version 15.7 or later
- Python installation – version 2.6, 2.7, or 3.1 built-in threaded mod
- Open Client SDK – version 15.7 or later

> **i Note**
>
> For information about platform support, see the *Software Developers Kit and Open Server Installation Guide* for your platform.

1.3 Extension Module for Python Installation Types

The extension module for Python is a component you can install through the SAP Installer.

The extension module for Python is an optional installation component when you choose Custom as the installation type. If the installation type you choose is Typical or Full, the extension module is installed by default. For complete installation and configuration instructions, see the *Software Developers Kit and Open Server Installation Guide* for your platform.

1.4 Extension Module for Python Configuration Overview

Complete basic configuration tasks for a Python application to make a connection and execute commands.

- Python module search path
- The name and address of the target server
- Security and directory services
- Runtime configuration through the `ocs.cfg` file

Related Information

Python Module Search Path [page 4]
Target Server Name and Address [page 5]
Security and Directory Services [page 5]
Runtime Configuration [page 6]

1.4.1 Python Module Search Path

Python searches for an imported module in the list of directories given by the Python variable `sys.path`.

This variable is initialized from the directory containing the application, and in the list of directories specified by the environment variable `<PYTHONPATH>`, which uses the same syntax as the shell variable `<PATH>`, that is, a list of directory names. If you have not set `<PYTHONPATH>`, or if the file is not found, the search continues in an installation-dependent default path.

To use the SAP Adaptive Server Enterprise extension module for Python in an application, set either <PYTHONPATH>, or the Python variable sys.path to one of the following directory paths (these directories are the default directories where the different versions of the SAP Adaptive Server Python extension module are installed):

Platform	Default Installation Path	Python Version
Windows	%SYBASE%\%SYBASE_OCS%\python\python26_64\dll	2.6
	%SYBASE%\%SYBASE_OCS%\python\python27_64\dll	2.7
	%SYBASE%\%SYBASE_OCS%\python\python31_64\dll	3.1
All other platforms	$SYBASE/$SYBASE_OCS/python/python26_64r/lib	2.6, 2.7
	$SYBASE/$SYBASE_OCS/python/python31_64r/lib	3.1

1.4.2 Target Server Name and Address

When the Python application connects to the SAP ASE Server, you can get the name of the target server from one of these sources.

1. The server name, if specified in the connect method.
2. The <DSQUERY> environment variable, if the application does not specify the target server in the connect method.
3. The <SYBASE> environment variable, if <DSQUERY> is not set.

You can get the address of the target server from the directory service or from the platform-dependent interfaces file. Create a server entry in the interfaces file or the LDAP directory service. See the *Open Client and Open Server Configuration Guide* for your platform.

1.4.3 Security and Directory Services

Configure the directory driver or security driver in the libtcl.cfg file in these sections.

- Directory driver in the [DIRECTORY] section.
- Security driver in the [SECURITY] section.

See the *Open Client and Open Server Configuration Guide* for your platform.

1.4.4 Runtime Configuration

Use the runtime configuration file `ocs.cfg` to set these values.

- Property values
- Server option values
- Server capabilities
- Debugging options

See *Using the Open Client and Open Server Runtime Configuration File* in the *Open Client Client-Library/C Reference Manual* for information about the file syntax and the properties that can be set in the file.

1.5 Developing Python Applications

Overview of how to write Python applications using the sybpydb module.

Related Information

Thread Safety [page 6]
Parameter Style [page 7]
Loading the Extension Module for Python [page 7]
Establish and Close a Connection to SAP Adaptive Server Enterprise using DSN-style Connection String Properties [page 7]
Bulk Copy Support [page 11]
Accessing and Updating Data Using Python [page 11]
Passing Input and Output Parameters to Stored Procedures [page 12]
Compute Rows Processing [page 13]
Parameter Support for Dynamic Statements and Stored Procedures [page 13]

1.5.1 Thread Safety

Threads may share the module. However, a connection, its attributes, and any objects created using the connection cannot be shared without a locking mechanism.

A cursor is an example of an object created from a connection. To use a connection among multiple threads, wrap the connection (its attributes and created objects) using a semaphore to implement resource locking.

1.5.2 Parameter Style

The module supports question mark style parameter marker formatting.

1.5.3 Loading the Extension Module for Python

Use the import statement to load the extension module for Python.

Procedure

To use the sybpydb module from a Python script, import the sybpydb module:

```
import sybpydb
```

1.5.4 Establish and Close a Connection to SAP Adaptive Server Enterprise using DSN-style Connection String Properties

Use the `connect` method to open a database connection. The method supports DSN-style connection properties.

The method accepts the following keyword arguments:

- `user` – the user login name that the connection uses to log in to a server.
- `password` – the password that a connection uses when logging in to a server.
- `servername` – defines the SAP ASE server name to which client programs connect. If you do not specify `servername`, the DSQUERY environment variable defines the Adaptive Server name.
- `dsn` – the data source name. The data source name is a semicolon-separated string of name=value parts:
  - Name – a case-insensitive value that can be delimited by an equal sign (=) or semicolon (;). An attribute can have multiple synonyms. For example, `server` and `servername` refer to the same attribute.
  - Equals sign (=) – indicates the start of the value to be assigned to the Name. If there is no equals sign, the Name is assumed to be of Boolean type with a value of true.
  - Value – a string that is terminated by a semicolon (;). Use a backslash (\) if a semicolon or another back slash is present in the value. Values can be of type Boolean, integer, or string. Valid values for Boolean types are true, false, on, off, 1, and 0.

    > **i Note**
    >
    > If a Boolean name is present without a value, set the Boolean type to true.

  For example:

```
sybpydb.connect(user='name', password='password string',
```

```
                                dsn='servername=Sybase;timeout=10')
```

Related Information

Valid Attribute Names and Values [page 8]

1.5.4.1 Valid Attribute Names and Values

The table lists the valid attribute names and values for the `dsn` keyword argument.

Name	Description	Value
`ANSINull`	Determines whether evaluation of NULL-valued operands in SQL equality (=) or inequality (!=) comparisons is ANSI-compliant. If the value is true, Adaptive Server enforces the ANSI behavior that `<= NULL>` and `<is NULL>` are not equivalent. In standard Transact-SQL, `<= NULL>` and `<is NULL>` are considered to be equivalent. This option affects `<> NULL>` and `<is not NULL>` behavior in a similar fashion.	Boolean value. The default is false.
`BulkLogin`	Determines whether a connection is enabled to perform a bulk-copy operation.	Boolean value. The default is false.
`ChainXacts`	If true, SAP ASE uses chained transaction behavior. SAP ASE implicitly executes a begin transaction before any of these statements: `delete`, `fetch`, `insert`, `open`, `select`, and `update`. You must still explicitly end or roll back the transaction. If false, each server command is considered to be a distinct transaction. If an application needs to combine multiple statements as one big transaction, such as `update`, `insert`, or `delete`, it must specify explicit begin transaction statements paired with commit or rollback statements.	Boolean value. The default is false.
`Charset`	Specifies the `charset` to be used on this connection.	String value.
`Confidentiality`	Whether data encryption service is performed on the connection.	Boolean value. The default is false.

Name	Description	Value
`CredentialDelegation`	Determines whether to allow the server to connect to a second server with the user's delegated credentials.	Boolean value. The default is false.
`DetectReplay`	Determines whether the connection's security mechanism detects replayed transmissions.	Boolean value. The default is false.
`DetectOutOfSequence`	Determines whether the connection's security mechanism detects transmissions that arrive out of sequence.	Boolean value. The default is false.
`HostName`	Specifies, in the value section, the host name that will be stored in the SAP ASE`sysprocesses` table . If no `HostName` is specified, Python sends an empty string for the `HostName`.	String value.
`Integrity`	Determines whether the connection's security mechanism performs data integrity checking.	Boolean value. The default is false.
`Interfaces`	The path and name of the interfaces file.	String value.
`Keytab`	The name and path to the file from which a connection's security mechanism reads the security key to go with the `<username>` value.	String value. The default is NULL, that is, the user must have established credentials before connecting.
`Locale`	Determines which language and character set to use for messages, datatype conversions, and datetime formats.	String value.
`Language`	Determines which language set to use for messages, datatype conversions, and datetime formats.	String value.
`LoginTimeout`	Specifies the login timeout value.	Integer value.
`MaxConnect`	Specifies the maximum number of simultaneously open connections that a context may have.	Integer value. Default value is 25. Negative and zero values are not allowed.
`MutualAuthentication`	Determines whether the server is required to authenticate itself to the client.	Boolean value. The default is false.
`NetworkAuthentication`	Determines whether the connection's security mechanism performs network-based user authentication.	Boolean value. The default is false.

Name	Description	Value
`PacketSize`	Specifies the TDS packet size.	Integer value.
`Password`	Specifies the password used to log in to the server.	String value.
`PasswordEncryption`	Determines whether the connection uses asymmetrical password encryption.	Boolean value. The default is false.
`SecurityMechanism`	Specifies the name of the network security mechanism that performs security services for the connection.	String value. The default value depends on security driver configuration.
`Server` `Servername`	Specifies the name of the server to which you are connected.	String value.
`ServerPrincipalName`	Specifies the network security principal name for the server to which a connection is opened.	String value. The default is NULL, which means that the connection assumes the server principal name is the same as its `<ServerName>` value.
`ScriptName`	Specifies the chosen name of the top-level Python script that drives the application. This name appears in the sysprocesses table as the application name. Absence of this value gives a default application name of "sybpydb". This value can be as many as 256 characters.	String value.
`Keepalive`	Determines whether to use the KEEPALIVE option.	Boolean value. The default is true.
`Timeout`	Specifies the connection timeout value.	Integer value.

`UID`
`User`
`Username`
	Specifies the name used to log in to the server.	String value.

1.5.5 Bulk Copy Support

Support for a bulk copy operation is an extension to the Python DBAPI. It enables you to bulk copy rows.

An example for a bulk copy in operation:

```
import syDbpydb
conn = sybpydb.connect(dsn="user=john;bulklogin=true;chainxacts=off")
cur = conn.cursor()
cur.execute("create table mytable (id int identity, name varchar(20))")
cur.close()
blk = conn.blkcursor()
blk.copy("mytable", direction="in", )
blk.rowxfer(["Mark"])
blk.rowxfer(["Leanne"])
blk.rowxfer(["Stanley"])
blk.done()
blk.close()
conn.close()
```

1.5.6 Accessing and Updating Data Using Python

After a connection is established, use a cursor object to manage the context of a fetch operation.

A cursor object provides access to methods to prepare and execute queries, and fetch rows from a result set. The cursor object is obtained from the connection object using the `cursor` method. This example shows how an application accesses and updates data:

```
import sybpydb
   #Create a connection.
   conn = sybpydb.connect(user='sa')
   # Create a cursor object.
   cur = conn.cursor()
   cur.execute("drop table footab")
   cur.execute("create table footab ( id integer, first char(20) null, last
char(50) null)")
   cur.execute("insert into footab values( ?, ?, ? )", (1, "John", "Doe"))
   cur.execute("select * from footab")
   rows = cur.fetchall()
   for row in rows:
       print "-" * 55
       for col in range (len(row)):
          print "%s" % (row[col]),
   #Close the cursor object
   cur.close()
   #Close the connection
   conn.close()
```

1.5.7 Passing Input and Output Parameters to Stored Procedures

Starting with 15.7 ESD#3, the SAP Adaptive Server Enterprise extension module for Python supports passing input and output parameters to stored procedures.

Use the `callproc()` method of the Cursor object to call a stored procedure. If there is an error in executing the stored procedure, `callproc()` throws an exception and you can retrieve the status value using the `proc_status` attribute. This support is an extension to the Python DBAPI specification.

This is a sample Python application with multiple row results:

```
import sybpydb
#Create a connection.
conn = sybpydb.connect(user='sa')
# Create a cursor object.
cur = conn.cursor()
# Call the stored procedure
try:
    cur.callproc('myproc')
    continue = True
    while(continue == True):
        row = cur.fetchall()
        continue = cur.nextset()
except sybpydb.Error:
    print("Status=%d" % cur.proc_status)
```

To specify output parameters, the extension module provides the OutParam constructor. This support is an extension to the Python DBAPI specification. The `callproc()` method returns a list of all the parameters passed to the method. If there are output parameters, and no result sets generated from the store procedure, the list contains the modified output values as soon as `callproc()` completes. However, if there are result sets, the list does not contain modified output values until all result sets from the stored procedure have been retrieved using the `fetch*()` methods and a call to `nextset()` is made to check if there are any more result sets. The `nextset()` method must be called even if only one result set is expected.

This is a sample Python application with output parameters:

```
import sybpydb
#Create a connection.
conn = sybpydb.connect(user='sa')
# Create a cursor object.
cur = conn.cursor()
cur.execute("""
    create procedure myproc
    @int1 int,
    @int2 int output
    as
    begin
        select @int2 = @int1 * @int1
    end
    """)
int_in = 300
int_out = sybpydb.OutParam(int())
vals = cur.callproc('pyproc', (int_in, int_out))
print ("Status = %d" % cur.proc_status)
print ("int = %d" % vals[1])
cur.connection.commit()
# Remove the stored procedure
cur.execute("drop procedure myproc")
cur.close()
```

```
conn.close()
```

More examples of different output parameter types are available in the sample program `callproc.py`.

1.5.8 Compute Rows Processing

Starting with 15.7 ESD#3, the SAP Adaptive Server Enterprise extension module for Python supports compute rows processing.

An example is available in the sample program `compute.py`.

1.5.9 Parameter Support for Dynamic Statements and Stored Procedures

Starting with 15.7 ESD#4, the SAP Adaptive Server Enterprise extension module for Python supports decimal, money, and LOB as parameters for dynamic statements and stored procedures.

Decimal and money type parameters

The following is an example usage of decimal and money as parameters for a stored procedure:

```
cur.execute("""
    create procedure pyproc
    @m1 money,
    @m2 money output,
    @d1 decimal(5,3),
    @d2 decimal(5,3) output,
    as
    begin
        select @d2 = @d1
        select @m2 = @m1
    end
    """)
dec_in = decimal.Decimal('1.23')
dec_out = sybpydb.OutParam(decimal.Decimal('0.00'))
dec_in = decimal.Decimal('1.23')
dec_out = sybpydb.OutParam(decimal.Decimal('0.00'))
m_in = decimal.Decimal('9.87')
m_out = sybpydb.OutParam(decimal.Decimal('0.00'))
vals = cur.callproc('pyproc', (int_out, dec_in, dec_out, m_in, m_out))
print ("Status = %d" % cur.proc_status)
print ("decimal = %s" % vals[1])
print ("money= %s" % vals[3])
```

Support for date, time, datetime, and float parameters for stored procedures

The SAP ASE extension module for Python supports for date, time, datetime, and float parameters for stored procedures.

See `callproc.py` sample that demonstrates calling stored procedures with parameters of different datatypes including date, time, datetime, float, and integer. The sample also demonstrates the handling of output parameters.

1.6 Extension Module for Python API Reference

The sybpydb extension interface API.

Related Information

Module Interface Methods [page 14]
Module Interface Constants [page 15]
Connection Object Methods [page 15]
Cursor Object Methods [page 16]
Warning and Error Messages [page 17]
BulkCursor Object Constructor [page 19]

1.6.1 Module Interface Methods

The API used for module interface.

`connect`

Constructs a connection object representing a connection to a database. The method accepts these keyword arguments:

- `user` – the user login name that the connection uses to log on to a server.
- `password` – the password that a connection uses when logging on to a server.
- `servername` – defines the Adaptive Server name to which client programs connects. If this argument is not specified then the DSQUERY environment variable defines the Adaptive Server name.

```
sybpydb.connect(user='name', password='password string', servername='ase
servername')
```

1.6.2 Module Interface Constants

The constants used for the extension module interface.

`apilevel`

String constant stating the supported DB API level. The default value is 2.0.

```
class sybpydb.apilevel
```

`paramstyle`

String constant stating the type of parameter marker formatting expected by the interface. The value for this constant is `qmark`. The interface expects question mark style parameter formatting, for example:

```
'...WHERE name=?'
```

```
class sybpydb.paramstyle
```

`threadsafety`

Integer constant stating the level of thread safety that the interface supports. The `threadsafety` constant value for the module is 1, which indicates that the module can be shared but not the connections.

```
class sybpydb.threadsafety
```

1.6.3 Connection Object Methods

The APIs used for connection objects.

`close()`

Closes the connection to the server. The connection is unusable after the call, and raises an exception if any operation is attempted. The same applies to cursor objects attempting to access the connection.

```
connection.close()
```

`commit()`

Executes the command `commit`.

```
connection.commit()
```

`rollback()`

Executes the command `rollback`.

```
connection.rollback()
```

`cursor()`

This method constructs a new cursor object using the connection.

```
connection.cursor()
```

`messages()`

This is a Python list object to which the module appends tuples (exception class and exception object) for all messages that the module receives for this connection. An error on any cursor obtained from the same connection object, is appended to the messages attribute of the connection object of the cursor.

```
connection.messages()
```

Usage example:

```
try:
    cur.execute("select ...")
except sybpydb.Error:
    for err in cur.connection.messages:
        print("Exception %s, Value %s", % err[0], err[1])
```

1.6.4 Cursor Object Methods

The API used for cursor objects.

`close`

```
cursor.close()
```

`callproc`

Calls a stored database procedure with the given name. After fetching all the result sets and rows, use the `proc_status` attribute to check the status result of the stored procedure.

```
cursor.callproc()
```

`execute`

Prepares and executes a query.

```
cursor.execute()
```

`executemany`

Prepares a database operation and executes it against all parameter sequences found in the sequence `seq_of_parameters`.

```
cursor.executemany()
```

`fetch`

Fetches the next row of a query result set, returning a single sequence, or None when no more data is available.

```
cursor.fetch()
```

`fetchmany`

Fetches the next set of rows of a query result, returning a sequence of sequences, for example, a list of tuples. An empty sequence is returned when no rows are available.

```
cursor.fetchmany()
```

`fetchall`

Fetches all (remaining) rows of a query result, returning them as a sequence of sequences.

```
cursor.fetchall()
```

`description`

A read-only attribute describing the column information.

```
cursor.description()
```

`nextset`

Forces the cursor to skip to the next available set, discarding any remaining rows from the current set.

```
cursor.nextset()
```

`arraysize`

This read/write attribute specifies the number of rows to fetch at a time with `fetchmany()`. It defaults to 1 which indicates to fetch a single row at a time.

```
cursor.arraysize()
```

`proc_status`

Stored procedure return status results. Use this member after fetching all the result sets and rows, to check the status of the stored procedure.

```
cursor.proc_status
```

1.6.5 Warning and Error Messages

All error and warning information is available through exceptions and subclasses.

`Warning`

Exception raised for warnings. A subclass of the Python `StandardError` exception.

```
sybpydb.Warning
```

`Error`

Exception that is the base class of all other exceptions defined by sybpydb. `Error` is a subclass of the Python `StandardError` exception.

```
sybpydb.Error
```

`InterfaceError`

Exception raised for errors that are related to the database interface rather than the database itself. It is a subclass of `Error`.

```
sybpydb.InterfaceError
```

`DatabaseError`

Exception raised for errors that are related to the database. It is a subclass of `Error`.

```
sybpydb.DatabaseError
```

`DataError`

Exception raised for errors that are related to problems with the processed data. It is a subclass of `DatabaseError`.

```
sybpydb.DataError
```

`OperationalError`

Exception raised for errors that are related to problems to the operation of the database but are not necessarily under the control of the programmer. It is a subclass of `DatabaseError`.

```
sybpydb.OperationalError
```

`IntegrityError`

Exception raised when the relational integrity of the database is affected. It is a subclass of `DatabaseError`.

```
sybpydb.IntegrityError
```

`InternalError`

Exception raised when the database encounters an internal error. It is a subclass of `DatabaseError`.

```
sybpydb.InternalError
```

`ProgrammingError`

Exception raised for programming errors. It is a subclass of `DatabaseError`

```
sybpydb.ProgrammingError
```

`NotSupportedError`

Exception raised when an unsupported method or database API is used. It is a subclass of `DatabaseError`.

```
sybpydb.NotSupportedError
```

1.6.6 BulkCursor Object Constructor

Python extension module that provides a connection object to establish a connection to the database. The connection object includes a method for creating a new `BulkCursor` object, which manages the context of a bulk operation.

The `BulkCursor` object can be constructed only from a connection object that was established with a property marking the connection for use in a bulk operation.

Usage

```
import sybpydb
conn = sybpydb.connect(dsn="user=sa;bulk=true")cur = conn.cursor()
cur.execute("create table mytable (i int, c char(10))")
blk = conn.blkcursor()
```

close()

The `close()` method of the `BulkCursor` object closes a bulk operation. Once this method has been called, the bulk cursor object cannot be used. `close()` takes no arguments.

Usage

```
import sybpydb
conn = sybpydb.connect(dsn="user=sa;bulk=true")
blk = conn.blkcursor()
bblk.close()
```

copy()

The `copy()` method of the `BulkCursor` object initializes a bulk operation.

This method accepts the following arguments:

- `tablename` – a string specifying the name of the table for the bulk operation.
- `direction` – this is a keyword argument with these values: `<in>` and `<out>`.

Usage

```
import sybpydb
conn = sybpydb.connect(dsn="user=sa;bulk=true")
cur = conn.cursor()
cur.execute("create table mytable (i int, c char(10))")
blk = conn.blkcursor()
blk.copy("mytable", direction="out")
```

done()

The `done()` method of the `BulkCursor` object marks the completion of a bulk operation. To start another operation, call the `copy()` method.

Usage

```
import sybpydb
conn = sybpydb.connect(dsn="user=sa;bulk=true")
cur = conn.cursor()
cur.execute("create table mytable (i int, c char(10)
blk = conn.blkcursor()
blk.copy("mytable", direction="in")
...
blk.done()
blk.copy("mytable", direction="out")
...
blk.done()
blk.close()
```

2 Multiversion Concurrency Control

Multiversion concurrency control (MVCC) enables concurrent read-write operations, by allowing the server to lock rows for writing in one session while granting access to unaltered versions of these rows in another session, so that reads never wait for writes, and writes never wait for reads.

SAP ASE offers two types of MVCC:

- Regular MVCC – uses in-memory row versioning, where all inserts, updates, and deletes are performed in the in-memory row storage.
- On-disk multiversion concurrency control (on-disk MVCC) – stores row versions in a temporary database specified when the database is created or altered.

The two types use slightly different parameters to create or alter the database. In addition to the standard isolation levels offered by SAP ASE, SAP ASE uses three isolation levels specifically to enable multiversion concurrency control, which are not part of the standard SQL:

- Statement snapshot
- Transaction snapshot
- Readonly statement snapshot

Related Information

Connection Support in Multiversion Concurrency Control [page 21]
Scripting Drivers Support in Multiversion Concurrency Control [page 23]

2.1 Connection Support in Multiversion Concurrency Control

`ISOLATION_MODE` is a connection-level property that is used to enable the MVCC feature by converting ANSI isolation levels to snapshot isolation levels for MVCC- and on-disk MVCC-enabled tables.

> **i Note**
>
> SAP ASE provides the following techniques to migrate applications to use snapshot isolation seamlessly, and with minimal application changes. Perform these changes on the application in conjunction with any necessary server-side changes. If you do not set the `ISOLATION_MODE` property, existing applications default to run at ANSI isolation levels.

You can set the `ISOLATION_MODE` connection property for the client and map ANSI isolation levels to snapshot isolation levels. When `ISOLATION_MODE` is set to `snapshot`, the isolation level maps as:

- Isolation level 0 and 1 – mapping to the `statement snapshot` isolation level
- Isolation level 2 and 3 – mapping to the `transaction snapshot` isolation level

Once the `ISOLATION_MODE` connection property is set, the server overrides the `set transaction isolation` command from the connection or system procedure, which avoids having to make application changes to set isolation levels. For example, if a stored procedure sets the isolation level to ANSI isolation levels 0, 1, 2, or 3, but when using MVCC-enabled tables and the `ISOLATION_MODE` property is set, the ANSI isolation level for the stored procedure is converted to the `statement snapshot` or `transaction snapshot` isolation levels described above.

`ISOLATION_MODE` can have these values:

- `default` – the applications use the isolation levels specified by the application.
- `snapshot` – the application uses the snapshot isolation levels listed in the following table when it specifies traditional ANSI isolation levels for MVCC-enabled tables.
- `readonly_snapshot` – the application uses the snapshot isolation levels listed in the following table when it specifies traditional ANSI isolation levels for MVCC-enabled tables.

Session-level Isolation Level	Connection Isolation Mode set to DEFAULT	Connection Isolation Mode set to SNAPSHOT	Connection Isolation Mode set to READONLY_SNAPSHOT
`read uncommitted`	`read uncommitted`	`statement snapshot`	`readonly statement snapshot`
`read commited`	`read commited`	`statement snapshot`	`readonly statement snapshot`
`repeatable read`	`repeatable read`	`transaction snapshot`	`transaction snapshot`
`serializable read`	`serializable read`	`transaction snapshot`	`transaction snapshot`
`statement snapshot`	`statement snapshot`	`statement snapshot`	`statement snapshot`
`transaction snapshot`	`transaction snapshot`	`transaction snapshot`	`transaction snapshot`
`readonly statement snapshot`	`readonly statement snapshot`	`readonly statement snapshot`	`readonly statement snapshot`

2.2 Scripting Drivers Support in Multiversion Concurrency Control

The `IsolationMode=` connection string property is supported for the scripting drivers and maps to the `CS_OPT_ISOLATION_MODE` option. Valid values for this property are 0, 1, and 2.

3 Glossary

Glossary of term specific to scripting languages.

Client-Library	part of Open Client, a collection of routines for use in writing client applications. Client-Library is designed to accommodate cursors and other advanced features in the Sybase product line.
CS-Library	included with both the Open Client and Open Server products, a collection of utility routines that are useful to both Client-Library and Server-Library applications.
CT-Library	(CT-Lib API) is part of the Open Client suite and is required to let an scripting application connect to Adaptive Server.
Extension or module	the Python language can be extended by modules that are written in Python.
Python	is an interpreted, general-purpose high-level programming language. For more information, go to http://www.python.org .
thread	a path of execution through Open Server application and library code and the path's associated stack space, state information, and event handlers.
Transact-SQL	an enhanced version of the database language SQL. Applications can use Transact-SQL to communicate with Adaptive Server Enterprise.

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Gender-Related Language

We try not to use gender-specific word forms and formulations. As appropriate for context and readability, SAP may use masculine word forms to refer to all genders.

www.sap.com/contactsap

© 2018 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see https://www.sap.com/about/legal/trademark.html for additional trademark information and notices.

THE BEST RUN SAP