Python SQL Driver

4/15/2020 • 2 minutes to read • Edit Online

To install SQL driver for Python

You can connect to a SQL Database using Python on Windows, Linux, or macOS.

Getting Started

There are several python SQL drivers available. However, Microsoft places its testing efforts and its confidence in **pyodbc** driver. Choose a driver, and configure your development environment accordingly:

- Python SQL driver - pyodbc
- Python SQL driver - pymssql

Documentation

- Python documentation at Python.org

Community

- Azure Python Developer Center
- python.org Community

More Samples

- Create a Python web app using DJango and SQL Database in Azure Website
- Getting Started with Python on Windows
- Getting Started with Python on macOS
- Getting Started with Python on Ubuntu
- Getting Started with Python on Red Hat Enterprise Linux (RHEL)
- Getting Started with Python on SUSE Linux Enterprise Server (SLES)

Python SQL Driver - pyodbc

4/17/2020 • 2 minutes to read • Edit Online

To install SQL driver for Python

Getting Started

- Step 1: Configure development environment for pyodbc Python development
- Step 2: Create a SQL database for pyodbc Python development
- Step 3: Proof of concept connecting to SQL using pyodbc

Documentation

- pyodbc documentation

Step 1: Configure development environment for pyodbc Python development

4/17/2020 • 2 minutes to read • Edit Online

Windows

Connect to SQL Database by using Python - pyodbc on Windows:

1. **Download Python installer.**
   If your machine does not have Python, install it. Go to the Python download page and download the appropriate installer. For example, if you are on a 64-bit machine, download the Python 2.7 or 3.7 (x64) installer.

2. **Install Python**. Once the installer is downloaded, do the following steps: a. Double-click the file to start the installer. b. Select your language, and agree to the terms. c. Follow the instructions on the screen and Python should be installed on your computer. d. You can verify that Python is installed by going to `C:\Python27` or `C:\Python37` and run `python -V` or `py -V` (for 3.x)

3. **Install the Microsoft ODBC Driver for SQL Server on Windows**

4. **Open cmd.exe as an administrator**

5. **Install pyodbc using pip - Python package manager** (Replace `C:\Python27\Scripts` with your installed Python path)

```
> cd C:\Python27\Scripts
> pip install pyodbc
```

Linux

Connect to SQL Database by using Python - pyodbc:

1. **Open terminal**

2. **Install Microsoft ODBC Driver for SQL Server on Linux**

3. **Install pyodbc**

```
> sudo -H pip install pyodbc
```

Step 2: Create a SQL database for pyodbc Python development

4/17/2020 • 2 minutes to read • Edit Online

The samples in this section only work with the AdventureWorks schema, on either Microsoft SQL Server or Azure SQL Database.

Azure SQL Database

Create a SQL database in minutes using the Azure portal

Microsoft SQL Server

Microsoft SQL Server Samples on GitHub

Step 3: Proof of concept connecting to SQL using pyodbc

5/23/2020 • 2 minutes to read • Edit Online

This example is a proof of concept. The sample code is simplified for clarity, and doesn't necessarily represent best practices recommended by Microsoft.

To get started, run the following sample script. Create a file called test.py, and add each code snippet as you go.

```
> python test.py
```

Connect

```
import pyodbc
# Some other example server values are
# server = 'localhost\sqlexpress' # for a named instance
# server = 'myserver,port' # to specify an alternate port
server = 'tcp:myserver.database.windows.net'
database = 'mydb'
username = 'myusername'
password = 'mypassword'
cnxn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL
Server};SERVER='+server+';DATABASE='+database+';UID='+username+';PWD='+ password)
cursor = cnxn.cursor()
```

Run query

The cursor.executefunction can be used to retrieve a result set from a query against SQL Database. This function accepts a query and returns a result set, which can be iterated over with the use of cursor.fetchone()

```
#Sample select query
cursor.execute("SELECT @@version;")
row = cursor.fetchone()
while row:
    print(row[0])
    row = cursor.fetchone()
```

Insert a row

In this example, you see how to run an INSERT statement safely, and pass parameters. The parameters protect your application from SQL injection.

```
#Sample insert query
cursor.execute("""
INSERT INTO SalesLT.Product (Name, ProductNumber, StandardCost, ListPrice, SellStartDate)
VALUES (?,?,?,?,?)""",
'SQL Server Express New 20', 'SQLEXPRESS New 20', 0, 0, CURRENT_TIMESTAMP)
cnxn.commit()
row = cursor.fetchone()

while row:
    print('Inserted Product key is ' + str(row[0]))
    row = cursor.fetchone()
```

Azure Active Directory and the connection string

pyODBC uses the Microsoft ODBC driver for SQL Server. If your version of the ODBC driver is 17.1 or later, you can use the Azure Active Directory interactive mode of the ODBC driver through pyODBC. This interactive option works if Python and pyODBC permit the ODBC driver to display the dialog. The option is only available on Windows operating systems.

Example connection string for Azure Active Directory interactive authentication

The following example provides an ODBC connection string that specifies Azure Active Directory interactive authentication:

```
server=Server;database=Database;UID=UserName;Authentication=ActiveDirectoryInteractive;
```

For more information about the authentication options of the ODBC driver, see Using Azure Active Directory with the ODBC Driver.

Next steps

For more information, see the Python Developer Center.

Python SQL Driver - pymssql

4/17/2020 • 2 minutes to read • Edit Online

Install pyodbc Python Driver Install pymssql Python Driver

Getting Started

- Step 1: Configure development environment for pymssql Python development
- Step 2: Create a SQL database for pymssql Python development
- Step 3: Proof of concept connecting to SQL using pymssql

Documentation

- pymssql documentation

Step 1: Configure development environment for pymssql Python development

4/17/2020 • 2 minutes to read • Edit Online

You will need to configure your development environment with the prerequisites in order to develop an application using the Python Driver for SQL Server.

The Python SQL Drivers use the TDS protocol, which is enabled by default in SQL Server and Azure SQL Database. No additional configuration is required.

Windows

1. **Install Python runtime and pip package manager.**
   a. Go to python.org
   b. Click on the appropriate Windows installer msi link.
   c. Once downloaded run the msi to install Python runtime

2. **Download pymssql module** from here

   Make sure you choose the correct `whl` file. For example: If you are using Python 2.7 on a 64-bit machine choose `pymssql‑2.1.1‑cp27‑none‑win_amd64.whl`. Once you download the `whl` file, place it in the C:\Python27 folder.

3. **Open cmd.exe**

4. **Install pymssql module.**
   For example, if you are using Python 2.7 on a 64-bit machine:

```
> cd c:\Python27
> pip install pymssql‑2.1.1‑cp27‑none‑win_amd64.whl
```

Ubuntu Linux

1. **Install Python runtime and pip package manager.** Python comes pre-installed on most distributions of Ubuntu. If your machine does not have python installed, you can either download the source tarball from python.org and build locally, or you can use the package manager:

```
> sudo apt-get install python
```

2. **Open terminal**

3. **Install pymssql module and dependencies**

```
> sudo apt-get --assume-yes update
> sudo apt-get --assume-yes install freetds-dev freetds-bin
> sudo apt-get --assume-yes install python-dev python-pip
> sudo pip install pymssql
```

macOS

1. **Install Python runtime and pip package manager**
   a. Go to python.org
   b. Click on the appropriate macOS installer pkg link.
   c. Once downloaded run the pkg to install Python runtime

2. **Open terminal**

3. **Install Homebrew package manager**

```
> ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

4. **Install FreeTDS module**

```
> brew install FreeTDS
```

5. **Install pymssql module**

```
> sudo -H pip install pymssql
```

Step 2: Create a SQL database for pymssql Python development

4/17/2020 • 2 minutes to read • Edit Online

Install pyodbc Python Driver Install pymssql Python Driver

The samples in this section only work with the AdventureWorks schema, on either Microsoft SQL Server or Azure SQL Database.

Azure SQL Database

Create a SQL database in minutes using the Azure portal

Microsoft SQL Server

Microsoft SQL Server Samples on GitHub

Step 3: Proof of concept connecting to SQL using pymssql

4/17/2020 • 2 minutes to read • Edit Online

Install pyodbc Python Driver Install pymssql Python Driver

This example should be considered a proof of concept only. The sample code is simplified for clarity, and does not necessarily represent best practices recommended by Microsoft.

Step 1: Connect

The pymssql.connect function is used to connect to SQL Database.

```
    import pymssql
    conn = pymssql.connect(server='yourserver.database.windows.net', user='yourusername@yourserver',
password='yourpassword', database='AdventureWorks')
```

Step 2: Execute query

The cursor.execute function can be used to retrieve a result set from a query against SQL Database. This function essentially accepts any query and returns a result set, which can be iterated over with the use of cursor.fetchone().

```
    import pymssql
    conn = pymssql.connect(server='yourserver.database.windows.net', user='yourusername@yourserver',
password='yourpassword', database='AdventureWorks')
    cursor = conn.cursor()
    cursor.execute('SELECT c.CustomerID, c.CompanyName,COUNT(soh.SalesOrderID) AS OrderCount FROM
SalesLT.Customer AS c LEFT OUTER JOIN SalesLT.SalesOrderHeader AS soh ON c.CustomerID = soh.CustomerID GROUP
BY c.CustomerID, c.CompanyName ORDER BY OrderCount DESC;')
    row = cursor.fetchone()
    while row:
        print str(row[0]) + " " + str(row[1]) + " " + str(row[2])
        row = cursor.fetchone()
```

Step 3: Insert a row

In this example you will see how to execute an INSERT statement safely and pass parameters. Passing parameters as values protects your application from SQL injection.

```
    import pymssql
    conn = pymssql.connect(server='yourserver.database.windows.net', user='yourusername@yourserver',
password='yourpassword', database='AdventureWorks')
    cursor = conn.cursor()
    cursor.execute("INSERT SalesLT.Product (Name, ProductNumber, StandardCost, ListPrice, SellStartDate)
OUTPUT INSERTED.ProductID VALUES ('SQL Server Express', 'SQLEXPRESS', 0, 0, CURRENT_TIMESTAMP)")
    row = cursor.fetchone()
    while row:
        print "Inserted Product ID : " +str(row[0])
        row = cursor.fetchone()
    conn.commit()
    conn.close()
```

Step 4: Roll back a transaction

This code example demonstrates the use of transactions in which you:

- Begin a transaction
- Insert a row of data
- Roll back your transaction to undo the insert

```
    import pymssql
    conn = pymssql.connect(server='yourserver.database.windows.net', user='yourusername@yourserver',
password='yourpassword', database='AdventureWorks')
    cursor = conn.cursor()
    cursor.execute("BEGIN TRANSACTION")
    cursor.execute("INSERT SalesLT.Product (Name, ProductNumber, StandardCost, ListPrice, SellStartDate)
OUTPUT INSERTED.ProductID VALUES ('SQL Server Express New', 'SQLEXPRESS New', 0, 0, CURRENT_TIMESTAMP)")
    conn.rollback()
    conn.close()
```

Next steps

For more information, see the Python Developer Center.