Step 1.1 Install SQL Server

1. If you don't have SQL Server 2017 Developer (or above) installed, click **here** to download the SQL Server exe.
2. Run it to start the SQL installer.
3. Click **Basic** in *Select an installation type*.
4. Click **Accept** after you have read the license terms.
5. (Optional) if you need to, you can choose a custom installation location for SQL Server.
6. Click **Install** to proceed with the installation.

You now have SQL Server installed and running locally on your Windows computer! Check out the next section to continue installing prerequisites.

Step 1.2 Install Python

Download and run the installer **here**

Next, add Python to your path

1. Press start
2. Search for "Advanced System Settings"
3. Click on the "Environment Variables" button
4. Add the location of the Python27 folder to the PATH variable in System Variables. The following is a typical value for the PATH variable C:\Python27

You have succesfully installed Python on your machine!

Step 1.3 Install the ODBC Driver and SQL Command Line Utility for SQL Server

SQLCMD is a command line tool that enables you to connect to SQL Server and run queries.

1. Install the **ODBC Driver**.
2. Install the **SQL Server Command Line Utilities**.

After installing SQLCMD, you can connect to SQL Server using the following command from a CMD session:

Terminal

```
sqlcmd -S localhost -U sa -P your_password
1> # You're connected! Type your T-SQL statements here. Use the keyword 'GO' to
execute each batch of statements.
```

Copy

This how to run a basic inline query. The results will be printed to STDOUT.

Terminal

```
sqlcmd -S localhost -U sa -P yourpassword -Q "SELECT @@VERSION"
```

Copy

Results

```
------------------------------------------------------------
Microsoft SQL Server 2016 (RTM) - 13.0.1601.5 (X64)
  Apr 29 2016 23:23:58
  Copyright (c) Microsoft Corporation
  Developer Edition (64-bit)

1 rows(s) returned

Executed in 1 ns.
```

You have successfully installed SQL Server Command Line Utilities on your Windows machine!

Step 2.1 Install the Python driver for SQL Server

Terminal

```
pip install virtualenv #To create virtual environments to isolate package
installations between projects
virtualenv venv
venv\Scripts\activate
pip install pyodbc
```

Copy

Step 2.2 Create a database for your application

Connect to SQL Server using SQLCMD and execute the following statement to create a database called SampleDB.

Terminal

```
sqlcmd -S localhost -U sa -P your_password -Q "CREATE DATABASE SampleDB;"
```

Copy

Step 2.3 Create a Python app that connects to SQL Server and executes queries

Create a new folder for the sample

Terminal

```
mkdir SqlServerSample
cd SqlServerSample
```

Copy

Execute the T-SQL scripts below in the terminal with sqlcmd to a table and insert some row.

Terminal

```
sqlcmd -S localhost -U sa -P your_password -Q "USE DATABASE SampleDB; CREATE TABLE Employees (Id INT IDENTITY(1,1) NOT NULL PRIMARY KEY, Name NVARCHAR(50), Location NVARCHAR(50));"
sqlcmd -S localhost -U sa -P your_password -Q "USE DATABASE SampleDB; INSERT INTO Employees (Name, Location) VALUES (N'Jared', N'Australia'), (N'Nikita', N'India'), (N'Tom', N'Germany');"
```

Copy

Using your favorite text editor, create a new file called crud.py in the SqlServerSample folder. Paste the code below inside into the new file. This will insert, update, delete, and read a few rows.

Python

```
import pyodbc
server = 'localhost'
database = 'SampleDB'
username = 'sa'
password = 'your_password'
cnxn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server};SERVER='+server+';DATABASE='+database+';UID='+username+';PWD='+ password)
cursor = cnxn.cursor()

print ('Inserting a new row into table')
#Insert Query
tsql = "INSERT INTO Employees (Name, Location) VALUES (?,?);"
with cursor.execute(tsql,'Jake','United States'):
    print ('Successfully Inserted!')


#Update Query
print ('Updating Location for Nikita')
tsql = "UPDATE Employees SET Location = ? WHERE Name = ?"
with cursor.execute(tsql,'Sweden','Nikita'):
    print ('Successfully Updated!')


#Delete Query
print ('Deleting user Jared')
```

```
tsql = "DELETE FROM Employees WHERE Name = ?"
with cursor.execute(tsql,'Jared'):
    print ('Successfully Deleted!')


#Select Query
print ('Reading data from table')
tsql = "SELECT Name, Location FROM Employees;"
with cursor.execute(tsql):
    row = cursor.fetchone()
    while row:
        print (str(row[0]) + " " + str(row[1]))
        row = cursor.fetchone()
```

Copy

Run your Python script from the terminal.

Terminal

```
python crud.py
```

Copy

Results

```
Inserting a new row into table
Successfully Inserted!
Updating Location for Nikita
Successfully Updated!
Deleting user Jared
Successfully Deleted!
Reading data from table
Jake United States
```

Congratulations! You created your first Python app with SQL Server! Check out the next section to learn about how you can make your Python app faster with SQL Server's Columnstore feature.

Now that you have explored the basics, you are ready to see how you can make your app better with SQL Server. In this module we will show you a simple example of Columnstore Indexes and how they can improve data processing speeds. Columnstore Indexes can achieve up to 100x better performance on analytical workloads and up to 10x better data compression than traditional rowstore indexes.

Step 3.1 Create a new table with 5 million using sqlcmd

Terminal

```
sqlcmd -S localhost -U sa -P your_password -d SampleDB -t 60000 -Q "WITH a AS (SELECT
* FROM (VALUES(1),(2),(3),(4),(5),(6),(7),(8),(9),(10)) AS a(a))
```

```
SELECT TOP(5000000)
ROW_NUMBER() OVER (ORDER BY a.a) AS OrderItemId
,a.a + b.a + c.a + d.a + e.a + f.a + g.a + h.a AS OrderId
,a.a * 10 AS Price
,CONCAT(a.a, N' ', b.a, N' ', c.a, N' ', d.a, N' ', e.a, N' ', f.a, N' ', g.a, N' ',
h.a) AS ProductName
INTO Table_with_5M_rows
FROM a, a AS b, a AS c, a AS d, a AS e, a AS f, a AS g, a AS h;"
```
Copy

Step 3.2 Create a Python app that queries this tables and measures the time taken

Terminal

```
mkdir SqlServerColumnstoreSample
cd SqlServerColumnstoreSample
```
Copy

Using your favorite text editor, create a new file called columnstore.py in the SqlServerColumnstoreSample folder. Paste the following code inside it.

Python

```
import pyodbc
import datetime
server = 'localhost'
database = 'SampleDB'
username = 'sa'
password = 'your_password'
cnxn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL
Server};SERVER='+server+';DATABASE='+database+';UID='+username+';PWD='+ password)
cursor = cnxn.cursor()
tsql = "SELECT SUM(Price) as sum FROM Table_with_5M_rows"
a = datetime.now()
with cursor.execute(tsql):
  b = datetime.now()
  c = b - a
  for row in cursor:
    print ('Sum:', str(row[0]))
  print ('QueryTime:', c.microseconds, 'ms')
```
Copy

Step 3.3 Measure how long it takes to run the query

Run your Python script from the terminal.

Terminal

```
python columnstore.py
```

Copy

Results

```
Sum: 50000000
QueryTime: 363ms
```

Step 3.4 Add a columnstore index to your table.

Terminal

```
sqlcmd -S localhost -U sa -P your_password -d SampleDB -Q "CREATE CLUSTERED
COLUMNSTORE INDEX Columnstoreindex ON Table_with_5M_rows;"
```

Copy

Step 3.5 Measure how long it takes to run the query with a columnstore index

Terminal

```
python columnstore.py
```

Copy

Results

```
Sum: 50000000
QueryTime: 5ms
```

Congratulations! You just made your Python app faster using Columnstore Indexes!

End example 1

Example 2: Steps to Set Up Python SQL Server Integration using Pyodbc

This method relies on "**pyodbc**" library to set up the Python SQL Server Integration. The pyodbc library provides Python developers with easy access to ODBC (Open Database Connectivity) databases. Therefore, you can implement the method given in this section to set up Python ODBC integrations with any platforms such as MS Access, MySQL, IBM Db2, etc.

The following steps will allow you to easily set up your Python SQL Server Integration:

- Step 1**:** Establish the SQL Server Connection
- Step 2: Run an SQL Query
- Step 3: Extract Query Results to Python
- Step 4: Apply Modifications in SQL Server
- Step 5: Automate the Python SQL Server Functioning

Step 1: Establish the Python SQL Server Connection

The first step of setting up the Python SQL Server Integration requires you to build a connection between Python and the SQL server using the **pyodbc.connect** function and pass a connection string. The Python MsSQL Connection string will define the DBMS Driver, connection settings, the Server, and a specific Database.

Now, for instance, you wish to connect to server USXXX00345,67800 and a database DB02 using the SQL Server Native Client 11.0.

There are 2 ways to establish this Python SQL Server connection:

- **Approach 1 to Setup Python SQL Server Connection**: You can depend on a trusted internal connection using the following code:

```
cnxn_str = ("Driver={SQL Server Native Client 11.0};"
            "Server=USXXX00345,67800;"
            "Database=DB02;"
            "Trusted_Connection=yes;")
cnxn = pyodbc.connect(cnxn_str)
```

- **Approach 2 to Setup Python SQL Server Connection**: You don't have a trusted internal connection and wish to set up the required SQL Server connection using SQL Server Management Studio (SSMS). This will require you to enter your username (say, Alex) and password(Alex123) as shown in the following code:

```
cnxn_str = ("Driver={SQL Server Native Client 11.0};"
            "Server=USXXX00345,67800;"
            "Database=DB02;"
            "UID=Alex;"
            "PWD=Alex123;")
cnxn = pyodbc.connect(cnxn_str)
```

Now, as your Python database connection is in place, you can perform SQL queries via Python.

Step 2: Run an SQL Query

Now, every query that you will perform on the SQL Server will involve a cursor initialization and query execution sequence. Moreover, any changes made inside the SQL Server must also reflect in Python (which is covered in Step3 of Python MS SQL Server Integration).

You can initialize a cursor via:

```
cursor = cnxn.cursor()
```

Now, if you wish to perform a query, call this cursor object. For example, the following query will select the top 100 rows from a SQL table name associates:

```
cursor.execute("SELECT TOP(100) * FROM associates")
```

This query will give you the desired results, however, no data will be returned to Python. To ensure that your SQL changes are reflected in Python, move on to the next step of Python SQL Server Integration.

Step 3: Extract Query Results to Python

To extract your data from SQL Server into Python, you will need the **Pandas** library. Pandas contain the "**read_sql**" function which is useful for reading data from SQL into Python. The read_sql requires a query and also the connection instance cnxn to extract the given data as follows:

```
data = pd.read_sql("SELECT TOP(100) * FROM associates", cnxn)
```

This will return a data frame consisting of the top 100 rows from your associates table.

Step 4: Apply Modifications in SQL Server

Next, if you wish to change the SQL data, you must add another step to the query execution process. This is because when you execute SQL queries, the changes are stored in a temporary space instead of directly modifying your stored data.

To make such modifications permanent, you have to **commit** them. For instance, if you wish to merge the firstName and lastName columns, generate a fullName column using the below code:

```
cursor = cnxn.cursor()
# first alter the table, adding a column
cursor.execute("ALTER TABLE associates " +
               "ADD fullName VARCHAR(20)")
# now update that column to contain firstName + lastName
cursor.execute("UPDATE associate " +
               "SET fullName = firstName + " " + lastName")
```

Even after executing this code, you won't find any **fullName** column in your **associate** database. You need to commit the above changes and make them permanent via the following command:

```
cnxn.commit()
```

Step 5: Automate the Python SQL Server Functioning

The above steps discussed how you can modify your SQL data and extract it to Python or, alternatively, you can first extract the data to Python and then perform the manipulations. Once this setup is ready, you can use the Python SQL Server Integration to automate a multitude of tasks.

For example, you may need to perform daily reporting which involves querying the newest batch of data within the SQL server, calculating basic statistics, and sending the results via email to the management. You can automate this lengthy process by leveraging the Python SQL Server Integration as follows:

```
# imports for SQL data part
import pyodbc
```

```
from datetime import datetime, timedelta
import pandas as pd

# imports for sending email
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import smtplib

date = datetime.today() - timedelta(days=7)  # get the date 7 days ago

date = date.strftime("%Y-%m-%d")  # convert to format yyyy-mm-dd

cnxn = pyodbc.connect(cnxn_str)  # initialise connection (assume we have already defined cnxn_str)

# build up our query string
query = ("SELECT * FROM associates "
        f"WHERE joinDate > '{date}'")

# execute the query and read to a dataframe in Python
data = pd.read_sql(query, cnxn)

del cnxn  # close the connection

# make a few calculations
mean_payment = data['payment'].mean()
std_payment = data['payment'].std()

# get max payment and product details
max_vals = data[['product', 'payment']].sort_values(by=['payment'], ascending=False).iloc[0]

# write an email message
txt = (f"Customer reporting for period {date} - {datetime.today().strftime('%Y-%m-%d')}.nn"
       f"Mean payment amounts received: {mean_payment}n"
       f"Standard deviation of payment amounts: {std_payments}n"
       f"Highest payment amount of {max_vals['payment']} "
       f"received from {max_vals['product']} product.")

# we will built the message using the email library and send using smtplib
msg = MIMEMultipart()
msg['Subject'] = "Automated customer report"  # set email subject
msg.attach(MIMEText(txt))  # add text contents

# we will send via outlook, first we initialise connection to mail server
smtp = smtplib.SMTP('smtp-mail.outlook.com', '587')
smtp.ehlo()  # say hello to the server
smtp.starttls()  # we will communicate using TLS encryption

# login to outlook server, using generic email and password
```

```
smtp.login('Alex@outlook.com', 'Alex123')

# send email to our boss
smtp.sendmail('Alex@outlook.com', 'boss@outlook.com', msg.as_string())

# finally, disconnect from the mail server
smtp.quit()
```

This code will now extract the prior week's data, calculate your key metrics, and send a summary on your boss's email id.

End example 2

Say Squeeze then say SQL – why is it Ess Q El and not Skwel?

Reference

1. Python SQL Server Integration using Pyodbc : 5 easy steps (hevodata.com)
2. Python driver for SQL Server - Python driver for SQL Server | Microsoft Docs
3. GitHub - mkleehammer/pyodbc: Python ODBC bridge
4. Connect Python and SQL Server (tutorialgateway.org)
5. Connecting to Microsoft SQL server using Python - Stack Overflow